Arithmetic of Large Integers

In[21]:=
```
50!
```

Out[21]=
$$30\,414\,093\,201\,713\,378\,043\,612\,608\,166\,064\,768\,844\,377\,641\,568\,960\,512\,000\,000\,000\,000$$

In[22]:=
```
50!
---
2^100
```

Out[22]=
$$\frac{216\,105\,129\,892\,080\,882\,169\,214\,875\,191\,192\,738\,017\,616\,943\,359\,375}{9\,007\,199\,254\,740\,992}$$

In[23]:=
```
GCD[50!, 2^100]
```

Out[23]=
$$140\,737\,488\,355\,328$$

In[24]:=
```
FactorInteger[50!]
```

Out[24]=
$$\begin{pmatrix} 2 & 47 \\ 3 & 22 \\ 5 & 12 \\ 7 & 8 \\ 11 & 4 \\ 13 & 3 \\ 17 & 2 \\ 19 & 2 \\ 23 & 2 \\ 29 & 1 \\ 31 & 1 \\ 37 & 1 \\ 41 & 1 \\ 43 & 1 \\ 47 & 1 \end{pmatrix}$$

In[25]:=
```
2^256 + 1
```

Out[25]=
$$115\,792\,089\,237\,316\,195\,423\,570\,985\,008\,687\,907\,853\,269\,984\,665\,640\,564\,039\,457\,584\,007\,913\,129\,639\,937$$

In[26]:=
```
FactorInteger[2^256 + 1]
```

Out[26]=
$$\begin{pmatrix} 1\,238\,926\,361\,552\,897 & 1 \\ 93\,461\,639\,715\,357\,977\,769\,163\,558\,199\,606\,896\,584\,051\,237\,541\,638\,188\,580\,280\,321 & 1 \end{pmatrix}$$

In[27]:=
```
p = Random[Integer, {1, 10^1000}]
```

Out[27]=
$$5\,732\,633\,648\,444\,029\,959\,171\,112\,534\,135\,601\,537\,904\,527\,274\,424\,900\,382\,532\,001\,737\,941\,474\,619\,400\,523\,520\,462\,230\,330\,041 \cdots$$
$$125\,536\,677\,017\,734\,855\,696\,154\,228\,338\,013\,057\,234\,856\,311\,747\,239\,817\,067\,594\,543\,153\,099\,451\,342\,467\,908\,368\,438\,222\,191 \cdots$$
$$627\,068\,172\,197\,302\,715\,770\,028\,566\,855\,392\,342\,976\,305\,724\,205\,594\,772\,269\,515\,365\,592\,152\,841\,739\,667\,322\,895\,223\,024\,545 \cdots$$
$$931\,314\,017\,916\,679\,691\,952\,991\,750\,781\,258\,479\,158\,612\,316\,266\,439\,307\,568\,075\,105\,060\,765\,650\,029\,143\,393\,773\,806\,384\,838 \cdots$$
$$054\,113\,900\,232\,630\,543\,768\,575\,297\,229\,123\,646\,861\,592\,821\,903\,714\,826\,429\,093\,696\,306\,882\,866\,700\,765\,055\,250\,127\,946\,457 \cdots$$
$$077\,010\,917\,766\,830\,868\,844\,501\,764\,009\,407\,577\,366\,542\,471\,612\,721\,987\,372\,263\,059\,710\,647\,889\,656\,277\,818\,552\,220\,083\,011 \cdots$$
$$012\,443\,819\,051\,093\,799\,152\,622\,228\,227\,468\,804\,965\,725\,237\,086\,229\,146\,631\,197\,646\,069\,067\,216\,158\,731\,032\,894\,437\,169\,807 \cdots$$
$$273\,359\,707\,925\,736\,247\,280\,428\,972\,993\,013\,756\,656\,621\,288\,493\,097\,888\,202\,673\,076\,506\,482\,522\,570\,230\,411\,143\,623\,083\,375 \cdots$$
$$689\,967\,072\,678\,198\,299\,241\,685\,431\,292\,981\,759\,540\,833\,694\,545\,655\,684\,578\,302\,520\,267\,074\,775\,206\,768\,184\,557\,835\,920\,808 \cdots$$
$$565\,506\,538\,948\,737\,045\,985\,068\,313\,874\,334\,285\,231\,391\,055\,943\,208\,282\,058\,486\,335\,256\,284\,460\,460\,169\,312\,093\,906\,675\,913 \cdots$$
$$434\,377\,163\,339\,887\,396\,227\,651\,392\,427\,613\,751\,940\,796\,098\,619\,138\,230\,445\,298\,163\,341\,764$$

In[28]:=
```
FactorInteger[p]
```

Out[28]=
$Aborted

In[29]:=
```
PrimeQ[p]
```

Out[29]= False

Euclidean Algorithm

In[30]:=
```
Clear[gcd]
gcd[a_, b_] := gcd[|a|, |b|] /; a < 0 || b < 0
gcd[a_, b_] := gcd[b, a] /; a < b
gcd[a_, 0] := a
gcd[a_, b_] := gcd[b, Mod[a, b]]
```

In[35]:=
```
gcd[50!, 2^100]
```

Out[35]= 140 737 488 355 328

In[36]:=
```
GCD[50!, 2^100]
```

Out[36]= 140 737 488 355 328

In[37]:=
```
q = Random[Integer, {1, 10^1000}]
```

Out[37]= 970 963 574 550 034 971 479 320 522 180 098 774 644 844 555 258 534 666 014 431 037 461 961 913 280 164 165 977 122 765 152 ⋱
976 734 384 564 920 487 372 013 168 674 712 292 351 627 286 229 503 182 375 530 733 388 863 638 323 172 305 664 558 361 242 ⋱
868 730 754 294 684 816 042 820 013 882 414 394 681 495 566 844 249 351 278 236 812 255 289 985 516 525 873 739 272 675 944 ⋱
121 352 441 605 973 581 029 525 487 971 281 610 734 448 521 028 001 365 571 928 845 565 943 005 126 001 307 306 083 520 312 ⋱
750 574 464 682 590 614 363 814 914 599 711 899 668 118 513 199 960 696 128 146 014 404 843 161 945 562 632 349 038 134 204 ⋱
036 567 187 287 004 346 822 052 369 444 522 524 046 968 213 338 282 220 162 178 195 046 841 113 578 365 872 971 039 179 492 ⋱
413 610 442 085 444 886 942 268 422 260 375 218 306 753 386 951 711 315 886 306 372 331 052 907 021 001 297 088 033 088 290 ⋱
975 616 587 217 670 685 859 101 605 890 418 035 234 527 098 209 382 382 317 841 826 681 723 303 589 485 106 890 015 968 509 ⋱
626 835 925 306 583 475 997 063 643 433 677 019 825 788 715 203 684 341 138 905 519 821 168 351 385 502 796 691 205 761 068 ⋱
138 173 051 262 595 069 348 961 979 392 742 559 689 643 017 703 480 729 429 468 795 334 464 859 416 031 678 832 259 502 401 ⋱
256 965 643 106 844 278 887 423 583 842 981 783 493 769 276 691 839 508 735 376 811 836 603

In[38]:=
```
Timing[gcd[p, q]]
```

Out[38]= {0.015, 1}

In[39]:=
```
Timing[GCD[p, q]]
```

Out[39]= {0., 1}

Extended Euclidean Algorithm

In[40]:=
```
Clear[extendedgcd]
extendedgcd[xx_, y_] := Module[{x, k, rule, r, q, X},
   x[0] = xx; x[1] = y; k = 0; rule = {}; gcdmatrix = {};
   While[Not[x[k + 1] == 0],
    k = k + 1;
    r[k] = Mod[x[k - 1], x[k]];
    q[k] = Quotient[x[k - 1], x[k]];
    x[k + 1] = r[k];
    AppendTo[rule, X[k + 1] → Collect[X[k - 1] - q[k] * X[k] /. rule, {X[0], X[1]}]];
    AppendTo[gcdmatrix, {k, "|", x[k - 1], "=", q[k], "*", x[k], "+", r[k]}];
   ];
   {x[k], Coefficient[X[k - 2] - q[k - 1] * X[k - 1] /. rule, {X[0], X[1]}]}
  ] /; IntegerQ[xx] && IntegerQ[y]
```

In[42]:=
```
extendedgcd[p, q]
```

Out[42]=
```
{1,
 {-144 937 165 637 129 356 044 016 270 957 112 753 216 047 798 221 641 832 784 002 751 950 454 772 045 138 203 278 337 383 ×.
    591 819 557 624 004 436 834 344 991 543 914 414 100 019 669 790 071 650 798 535 377 297 633 707 951 470 207 913 846 ×.
    328 275 411 882 222 735 054 257 088 787 702 330 606 455 857 121 933 678 026 532 022 159 718 407 073 112 840 496 561 ×.
    514 045 096 282 495 255 649 418 781 904 570 603 298 764 635 246 878 915 321 430 540 532 679 993 208 697 409 450 237 ×.
    298 753 854 236 881 260 316 080 320 015 207 472 742 582 372 386 521 399 300 999 141 928 162 129 147 946 303 435 525 ×.
    336 123 333 997 747 318 891 525 809 097 972 234 097 044 352 379 967 327 857 786 269 119 837 048 827 399 185 110 799 ×.
    213 542 241 756 730 465 277 827 921 810 326 077 804 665 441 297 682 285 916 085 216 106 705 824 011 518 062 035 218 ×.
    545 233 940 149 755 160 732 239 333 246 629 865 310 890 091 787 553 023 535 883 356 019 391 527 111 692 060 687 124 ×.
    635 801 889 023 088 126 892 938 754 463 955 601 138 158 786 941 796 497 113 106 112 599 388 381 437 530 758 594 615 ×.
    186 232 184 773 274 453 301 375 805 828 540 903 389 362 618 378 571 015 223 222 744 401 589 340 188 815 158 622 144 ×.
    376 009 424 004 401 538 830 102 670 584 433 895 773 164 861 969 700 657 590 060 899 658 134 721 767 219 263 127 272 ×.
    391 921 867 190 189 804 797 752 232 741 544 654 397,
  855 718 684 428 050 795 387 537 956 193 112 012 122 831 063 943 535 198 748 616 637 768 160 628 410 141 449 760 268 229 ×.
    409 625 720 295 392 805 609 813 041 706 650 642 517 086 966 664 166 127 774 853 301 608 721 369 564 172 126 724 606 079 ×.
    283 366 020 273 320 903 882 701 233 114 772 381 200 071 845 812 882 641 792 105 753 802 354 429 396 984 530 028 022 718 ×.
    497 803 775 075 162 455 067 935 512 905 564 240 653 608 322 785 268 410 163 580 977 300 116 549 752 910 100 153 402 034 ×.
    594 778 870 465 699 445 112 004 143 463 715 229 934 506 389 959 964 789 991 414 006 774 092 963 323 138 758 105 082 783 ×.
    209 765 812 409 591 200 770 913 965 492 837 234 163 659 177 751 368 319 896 401 353 467 924 569 185 620 479 503 275 822 ×.
    973 033 913 353 270 277 754 278 358 602 262 176 044 584 021 610 991 305 584 243 906 366 781 367 490 123 173 174 722 511 ×.
    635 612 049 002 609 548 481 123 659 331 285 258 880 324 054 018 323 223 717 442 672 487 506 261 656 158 039 073 592 621 ×.
    381 280 731 108 958 819 231 793 510 938 490 187 377 087 888 799 160 285 477 180 513 309 019 584 910 700 708 244 584 139 ×.
    259 842 159 722 599 183 198 455 339 536 679 090 454 548 975 344 723 740 900 562 870 806 422 120 734 856 847 678 334 734 ×.
    717 756 437 846 956 050 142 122 129 607 951 318 467 215 427 969 442 875 358 836 730 915 260 677 198 258 912 488 102 554 ×.
    939 175 503}}
```

In[43]:=
```
{g, {s, t}} = extendedgcd[50!, 2^100]
```

Out[43]=
```
{140 737 488 355 328, {2 300 207 045 532 015, -55 187 692 455 202 575 553 816 307 492 120 086 771 270 529 773 297}}
```

In[44]:=
```
s * 50! + t * 2^100
```

Out[44]=
```
140 737 488 355 328
```

In[45]:=
```
extendedgcd[Fibonacci[20], Fibonacci[19]]
```

Out[45]=
```
{1, {1597, -2584}}
```

In[46]:=
```
gcdmatrix
```

Out[46]=

1	|	6765	=	1	*	4181	+	2584
2	|	4181	=	1	*	2584	+	1597
3	|	2584	=	1	*	1597	+	987
4	|	1597	=	1	*	987	+	610
5	|	987	=	1	*	610	+	377
6	|	610	=	1	*	377	+	233
7	|	377	=	1	*	233	+	144
8	|	233	=	1	*	144	+	89
9	|	144	=	1	*	89	+	55
10	|	89	=	1	*	55	+	34
11	|	55	=	1	*	34	+	21
12	|	34	=	1	*	21	+	13
13	|	21	=	1	*	13	+	8
14	|	13	=	1	*	8	+	5
15	|	8	=	1	*	5	+	3
16	|	5	=	1	*	3	+	2
17	|	3	=	1	*	2	+	1
18	|	2	=	2	*	1	+	0

Fermat Test

In[47]:=
```
2^p
```

General::ovfl : Overflow occurred in computation. >>

Out[47]= Overflow[]

In[48]:=
```
p
```

Out[48]= 5 732 633 648 444 029 959 171 112 534 135 601 537 904 527 274 424 900 382 532 001 737 941 474 619 400 523 520 462 230 330 041 ⋱
125 536 677 017 734 855 696 154 228 338 013 057 234 856 311 747 239 817 067 594 543 153 099 451 342 467 908 368 438 222 191 ⋱
627 068 172 197 302 715 770 028 566 855 392 342 976 305 724 205 594 772 269 515 365 592 152 841 739 667 322 895 223 024 545 ⋱
931 314 017 916 679 691 952 991 750 781 258 479 158 612 316 266 439 307 568 075 105 060 765 650 029 143 393 773 806 384 838 ⋱
054 113 900 232 630 543 768 575 297 229 123 646 861 592 821 903 714 826 429 093 696 306 882 866 700 765 055 250 127 946 457 ⋱
077 010 917 766 830 868 844 501 764 009 407 577 366 542 471 612 721 987 372 263 059 710 647 889 656 277 818 552 220 083 011 ⋱
012 443 819 051 093 799 152 622 228 227 468 804 965 725 237 086 229 146 631 197 646 069 067 216 158 731 032 894 437 169 807 ⋱
273 359 707 925 736 247 280 428 972 993 013 756 656 621 288 493 097 888 202 673 076 506 482 522 570 230 411 143 623 083 375 ⋱
689 967 072 678 198 299 241 685 431 292 981 759 540 833 694 545 655 684 578 302 520 267 074 775 206 768 184 557 835 920 808 ⋱
565 506 538 948 737 045 985 068 313 874 334 285 231 391 055 943 208 282 058 486 335 256 284 460 460 169 312 093 906 675 913 ⋱
434 377 163 339 887 396 227 651 392 427 613 751 940 796 098 619 138 230 445 298 163 341 764

In[49]:=
```
Mod[2^p, p]
```

General::ovfl : Overflow occurred in computation. >>

Out[49]= Overflow[]

In[50]:=
```
PowerMod[2, p, p]
```

Out[50]= 5 031 607 935 165 256 386 156 702 587 694 704 901 471 540 720 627 008 923 909 828 067 563 296 958 575 803 588 872 789 549 243 ⋱
944 030 269 740 579 997 680 342 783 514 939 253 349 846 646 724 695 363 849 189 295 315 494 847 603 049 464 354 250 363 413 ⋱
840 015 419 593 359 280 275 937 227 708 413 296 368 541 339 165 744 169 254 635 226 515 691 151 231 448 407 651 910 737 936 ⋱
970 491 416 096 839 593 358 793 729 250 472 644 911 843 631 105 566 894 295 995 271 797 818 285 696 728 070 609 601 220 696 ⋱
982 047 457 642 541 495 028 008 796 824 403 258 095 115 618 414 142 687 539 385 606 404 918 738 875 490 011 408 230 750 207 ⋱
590 513 619 293 652 401 667 020 223 077 155 265 525 209 963 754 104 743 669 705 402 514 818 311 856 365 176 796 619 072 425 ⋱
650 687 667 626 158 878 712 253 767 148 238 351 104 635 335 657 349 310 570 915 446 369 102 316 854 167 328 449 608 478 479 ⋱
910 504 945 005 348 711 375 291 751 663 008 327 541 238 315 987 853 853 247 017 087 876 136 447 908 355 233 119 838 245 495 ⋱
401 001 882 093 196 700 230 043 659 415 220 321 260 196 253 554 855 183 636 808 004 280 249 963 267 261 032 512 421 464 781 ⋱
333 774 353 008 540 002 813 017 528 493 152 297 310 613 729 846 257 295 806 040 359 948 988 040 839 565 631 843 720 702 101 ⋱
086 433 738 832 756 654 297 408 361 828 323 653 018 846 950 025 389 094 751 960 421 845 660

In[51]:=
```
Timing[result1 = PowerMod[2, p - 1, p]]
```

Out[51]= {0.234,
5 382 120 791 804 643 172 663 907 560 915 153 219 688 033 997 525 954 653 220 914 902 752 385 788 988 163 554 667 509 939 ⋱
642 534 783 473 379 157 426 688 248 505 926 476 155 292 351 479 235 967 590 458 391 919 234 297 149 472 758 686 361 344 ⋱
292 802 733 541 795 895 330 998 022 982 897 281 902 819 672 423 531 685 669 470 762 075 296 053 921 996 485 557 865 273 ⋱
566 881 241 450 902 717 006 759 642 655 892 740 015 865 562 035 227 973 686 003 100 932 035 188 429 291 967 862 935 732 ⋱
191 703 802 767 518 080 678 937 586 019 398 292 047 026 763 452 478 354 220 158 928 756 984 239 651 355 900 802 788 127 ⋱
533 329 179 348 332 333 762 268 530 241 635 255 760 993 543 281 421 445 876 217 683 413 365 520 984 231 112 733 100 756 ⋱
321 497 674 419 577 718 331 565 743 338 626 338 932 437 997 687 853 578 035 180 286 371 789 228 601 056 546 219 084 766 ⋱
506 449 180 672 022 824 143 591 932 326 465 542 479 327 860 362 328 011 042 098 929 802 240 475 870 724 845 082 191 309 ⋱
485 239 292 822 131 730 664 435 545 484 477 385 697 499 735 864 545 354 101 040 400 514 974 050 255 434 107 555 262 273 ⋱
662 369 237 014 608 535 128 692 794 949 640 445 978 638 524 399 042 921 183 743 291 271 002 392 894 732 788 932 263 347 ⋱
602 636 250 649 867 471 968 813 689 007 260 405 451 086 322 025 262 529 877 127 968 702 479 821 524 322 263 662 598 629 ⋱
292 593 712}

In[52]:=
```
PrimeQ[p]
```

Out[52]= False

Fast Powers

■ Recursive Approach

In[53]:=
```
Clear[powermod]
powermod[a_, 0, p_] := 1
powermod[a_, n_, p_] := Mod[powermod[a, n/2, p]^2, p] /; EvenQ[n]
powermod[a_, n_, p_] := Mod[a * powermod[a, n - 1, p], p] /; OddQ[n]
$RecursionLimit = ∞;
```

In[58]:=
```
Timing[result2 = powermod[2, p - 1, p]]
```

Out[58]= {0.171,
5 382 120 791 804 643 172 663 907 560 915 153 219 688 033 997 525 954 653 220 914 902 752 385 788 988 163 554 667 509 939 ⋱
642 534 783 473 379 157 426 688 248 505 926 476 155 292 351 479 235 967 590 458 391 919 234 297 149 472 758 686 361 344 ⋱
292 802 733 541 795 895 330 998 022 982 897 281 902 819 672 423 531 685 669 470 762 075 296 053 921 996 485 557 865 273 ⋱
566 881 241 450 902 717 006 759 642 655 892 740 015 865 562 035 227 973 686 003 100 932 035 188 429 291 967 862 935 732 ⋱
191 703 802 767 518 080 678 937 586 019 398 292 047 026 763 452 478 354 220 158 928 756 984 239 651 355 900 802 788 127 ⋱
533 329 179 348 332 333 762 268 530 241 635 255 760 993 543 281 421 445 876 217 683 413 365 520 984 231 112 733 100 756 ⋱
321 497 674 419 577 718 331 565 743 338 626 338 932 437 997 687 853 578 035 180 286 371 789 228 601 056 546 219 084 766 ⋱
506 449 180 672 022 824 143 591 932 326 465 542 479 327 860 362 328 011 042 098 929 802 240 475 870 724 845 082 191 309 ⋱
485 239 292 822 131 730 664 435 545 484 477 385 697 499 735 864 545 354 101 040 400 514 974 050 255 434 107 555 262 273 ⋱
662 369 237 014 608 535 128 692 794 949 640 445 978 638 524 399 042 921 183 743 291 271 002 392 894 732 788 932 263 347 ⋱
602 636 250 649 867 471 968 813 689 007 260 405 451 086 322 025 262 529 877 127 968 702 479 821 524 322 263 662 598 629 ⋱
292 593 712}

In[59]:=
```
result1 - result2
```

Out[59]= 0

■ Iterative Approach

In[60]:=
```
digits = IntegerDigits[p - 1, 2]
```

Out[60]= {1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0,
1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0,
1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0,
1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,

1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1,
0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1,
0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1,
1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0,
0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0,
0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1,
0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0,
1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1,
1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0,
1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0,
1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1,
0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1,
1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0,
1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1,
1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0,
0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,
0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,

1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1,
0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1,
1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0,
1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0,
1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1,
0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1}

In[61]:= **Length[digits]**

Out[61]= 3322

In[62]:=
```
iterativepowermod[a_, n_, p_] := Module[{digits, result, power, k},
  digits = IntegerDigits[n, 2];
  result = 1; power = a;
  Do[If[digits[[Length[digits] - k + 1]] == 1, result = Mod[result * power, p]];
   power = Mod[power^2, p], {k, Length[digits]}];
  result
 ]
```

In[63]:= **Timing[result3 = iterativepowermod[2, p - 1, p]]**

Out[63]= {0.328,
5 382 120 791 804 643 172 663 907 560 915 153 219 688 033 997 525 954 653 220 914 902 752 385 788 988 163 554 667 509 939 ⋱
642 534 783 473 379 157 426 688 248 505 926 476 155 292 351 479 235 967 590 458 391 919 234 297 149 472 758 686 361 344 ⋱
292 802 733 541 795 895 330 998 022 982 897 281 902 819 672 423 531 685 669 470 762 075 296 053 921 996 485 557 865 273 ⋱
566 881 241 450 902 717 006 759 642 655 892 740 015 865 562 035 227 973 686 003 100 932 035 188 429 291 967 862 935 732 ⋱
191 703 802 767 518 080 678 937 586 019 398 292 047 026 763 452 478 354 220 158 928 756 984 239 651 355 900 802 788 127 ⋱
533 329 179 348 332 333 762 268 530 241 635 255 760 993 543 281 421 445 876 217 683 413 365 520 984 231 112 733 100 756 ⋱
321 497 674 419 577 718 331 565 743 338 626 338 932 437 997 687 853 578 035 180 286 371 789 228 601 056 546 219 084 766 ⋱
506 449 180 672 022 824 143 591 932 326 465 542 479 327 860 362 328 011 042 098 929 802 240 475 870 724 845 082 191 309 ⋱
485 239 292 822 131 730 664 435 545 484 477 385 697 499 735 864 545 354 101 040 400 514 974 050 255 434 107 555 262 273 ⋱
662 369 237 014 608 535 128 692 794 949 640 445 978 638 524 399 042 921 183 743 291 271 002 392 894 732 788 932 263 347 ⋱
602 636 250 649 867 471 968 813 689 007 260 405 451 086 322 025 262 529 877 127 968 702 479 821 524 322 263 662 598 629 ⋱
292 593 712}

In[64]:= **result3 - result1**

Out[64]= 0

Carmichael Numbers

In[65]:=
```
Clear[NextCarmichael]
NextCarmichael[n_] := Module[{ready = False, p = n, a},
  While[Not[ready],
   If[Not[PrimeQ[p]],
    list = Table[PowerMod[a, p, p] - a, {a, p - 1}];
    list = Union[list];
    If[list == {0}, ready = True, p = p + 1], p = p + 1]];
  p]
```

In[67]:=
```
(c = NextCarmichael[4]) // Timing
```

Out[67]= {0.375, 561}

In[68]:=
```
FactorInteger[c]
```

Out[68]=
$$\begin{pmatrix} 3 & 1 \\ 11 & 1 \\ 17 & 1 \end{pmatrix}$$

In[69]:=
```
(c = NextCarmichael[c + 1]) // Timing
```

Out[69]= {0.875, 1105}

In[70]:=
```
FactorInteger[c]
```

Out[70]=
$$\begin{pmatrix} 5 & 1 \\ 13 & 1 \\ 17 & 1 \end{pmatrix}$$

In[71]:=
```
(c = NextCarmichael[c + 1]) // Timing
```

Out[71]= {1.703, 1729}

In[72]:=
```
FactorInteger[c]
```

Out[72]=
$$\begin{pmatrix} 7 & 1 \\ 13 & 1 \\ 19 & 1 \end{pmatrix}$$

In[73]:=
```
(c = NextCarmichael[c + 1]) // Timing
```

Out[73]= {3.219, 2465}

In[74]:=
```
FactorInteger[c]
```

Out[74]=
$$\begin{pmatrix} 5 & 1 \\ 17 & 1 \\ 29 & 1 \end{pmatrix}$$

In[75]:=
```
(c = NextCarmichael[c + 1]) // Timing
```

Out[75]= {1.922, 2821}

In[76]:=
```
FactorInteger[c]
```

Out[76]=
$$\begin{pmatrix} 7 & 1 \\ 13 & 1 \\ 31 & 1 \end{pmatrix}$$

■ Structure of Carmichael Numbers

In[77]:=
```
Clear[FastNextCarmichael]
FastNextCarmichael[n_] := Module[{ready = False, p = n},
  While[Not[ready],
   If[PrimeQ[p], p = p + 1,
    factors = Transpose[FactorInteger[p]];
    If[Not[Union[factors[[2]]] == {1}] ||
      Not[IntegerQ[(p - 1) / Apply[LCM, factors[[1]] - 1]]], p = p + 1, ready = True]]
  ];
  p
 ]
```

In[79]:=
```
(c = FastNextCarmichael[4]) // Timing
```

Out[79]= {0.031, 561}

In[80]:=
```
(c = FastNextCarmichael[c + 1]) // Timing
```

Out[80]= {0.031, 1105}

In[81]:=
```
(c = FastNextCarmichael[c + 1]) // Timing
```

Out[81]= {0.031, 1729}

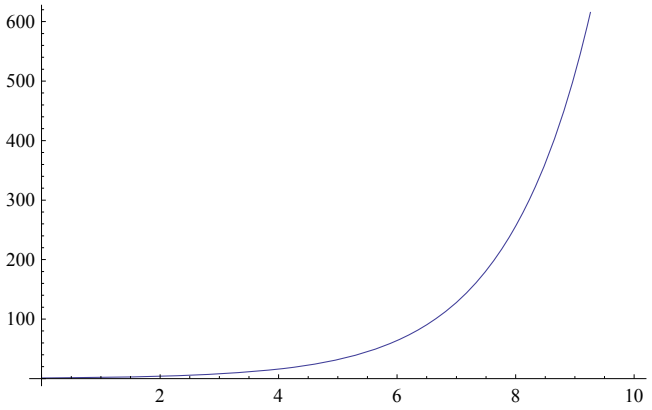
In[82]:=
```
(c = FastNextCarmichael[c + 1]) // Timing
```

Out[82]= {0.047, 2465}

Diffie-Hellman Key Exchange

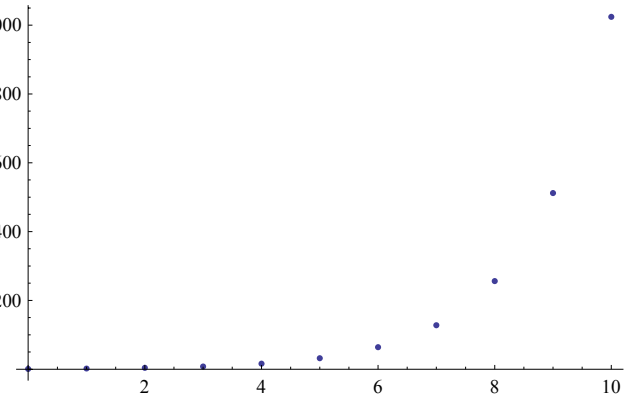
In[83]:=
```
Plot[2^x, {x, 0, 10}]
```

Out[83]=



In[84]:=
```
ListPlot[Table[{x, 2^x}, {x, 0, 10}], PlotStyle → PointSize[0.01]]
```

Out[84]=

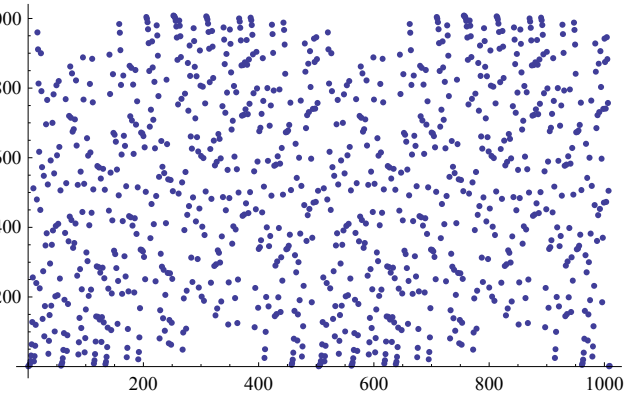


In[85]:=
```
NextPrime[1000]
```

Out[85]= 1009

In[86]:=
```
ListPlot[Table[{x, PowerMod[2, x, 1009]}, {x, 0, 1009}], PlotStyle → PointSize[0.01]]
```

Out[86]=

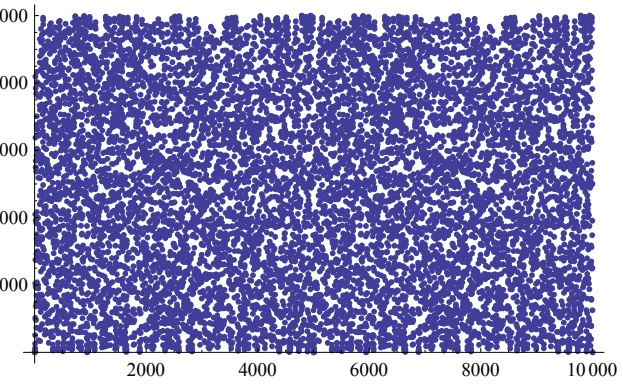


In[87]:=
```
NextPrime[10 000]
```

Out[87]= 10 007

In[88]:=
```
ListPlot[Table[{x, PowerMod[2, x, 10 007]}, {x, 0, 10 007}], PlotStyle → PointSize[0.01]]
```

Out[88]=



Modular Inverse

In[89]:=
```
φ = NextPrime[10^10]
```

Out[89]= 10 000 000 019

In[90]:= `e = NextPrime[10^5]`

Out[90]= 100 003

- **We search for the Inverse of e modulo φ**

In[91]:= `d = PowerMod[e, -1, φ]`

Out[91]= 5 357 139 296

- **Test:**

In[92]:= `Mod[e * d, φ]`

Out[92]= 1

- **The Extended Euclidiean Algorithm yields**

In[93]:= `{g, {s, t}} = extendedgcd[e, φ]`

Out[93]= {1, {−4 642 860 723, 46 430}}

In[94]:= `d == Mod[s, φ]`

Out[94]= True

The RSA Crypto System

- **Loading auxiliary functions...**
- **Generation of keys...**

In[95]:= `InitializeRSA`

- **public key**

In[96]:= `{e, m}`

Out[96]= {6 723 635 266 267 139 873 294 471 629 621 884 780 097,
6 214 707 391 762 720 857 355 655 434 330 870 654 747 875 220 690 846 000 396 605 221 612 423 877 887 783 939 283 756 920 ⋱
129 474 055 033 381 518 372 140 790 882 810 324 684 026 564 481 271 815 885 167 288 037 201 220 220 959 624 324 911 360 ⋱
936 259 960 748 551 164 615 082 888 025 669 361 171 292 874 708 580 040 380 769 315 305 816 922 206 661 935 622 939 325 ⋱
966 780 668 976 664 925 900 837 357 065 311 204 750 511 555 797 191 996 174 876 995 021 388 712 231 296 767 342 479 732 ⋱
398 074 607 316 268 091 404 637 681 272 934 843 871}

- **private key**

In[97]:= `d`

Out[97]= 2 828 548 488 868 288 182 968 692 346 268 559 402 286 955 940 097 770 155 290 794 089 571 878 560 037 583 208 590 440 677 412 ⋱
815 577 953 469 885 710 794 270 683 316 582 845 561 512 753 366 027 702 438 723 435 790 306 310 353 792 650 695 183 887 339 ⋱
840 604 480 895 880 565 743 671 393 193 124 967 814 798 109 167 874 750 514 501 792 088 280 257 909 725 999 792 092 612 637 ⋱
727 312 932 538 794 293 487 151 392 113 189 005 157 524 935 087 277 399 990 718 433 169 634 946 187 500 057 443 289 783 597 ⋱
810 020 670 929 883 323 784 759 745

encoding with RSA

In[98]:=
```
message = "This is my message"
```

Out[98]= This is my message

In[99]:=
```
result = EncodeRSA[message]
```

Out[99]= 8 906 194 969 641 276 075 023 548 572 467 116 958 404 955 733 552 539 026 837 272 197 083 731 032 575 448 141 394 762 206 426 ⋱
334 462 846 288 127 066 630 036 872 415 849 525 499 188 605 778 702 090 486 457 365 895 267 126 338 664 776 110 305 512 810 ⋱
537 228 042 144 172 873 207 504 727 724 859 371 186 297 440 757 528 737 540 213 594 089 444 681 158 502 229 453 927 483 837 ⋱
742 835 660 345 125 241 739 286 040 651 681 247 426 134 576 097 592 639 327 780 927 026 080 454 127 484 962 019 479 086 674 ⋱
803 366 010 143 465 221 082 881

In[100]:=
```
DecodeRSA[result]
```

Out[100]= This is my message

In[101]:=
```
DecodeRSA[result + 1]
```

Out[101]= □Y□□úýˆ□□t□φ□□ǐ□□o□ŨÝb□3γ□ļ+
ě□đ̃□□ȵn□T½□□Ñ¡□□ŜS□η□K□□ġé□□□□□□□ƒ□ς□□□□□Pŕn□□α□a□□□□□□
□ŭ□□□□Ê□6φ□ĄÃ□□G□Ľ□fŪ□ε□□□έ□Æ□□βąAsv□□Ěςű□□□□ÁΨ□U□□

■ another message

In[102]:=
```
result = EncodeRSA["What about this message?"]
```

Out[102]= 3 694 789 026 648 323 782 083 602 914 598 430 679 214 288 574 313 263 239 833 704 143 500 333 607 091 826 639 752 912 381 788 ⋱
288 236 133 112 897 680 347 776 913 286 002 189 187 056 619 102 384 438 449 751 508 259 594 440 750 029 473 350 742 141 562 ⋱
114 311 591 818 016 631 903 206 682 439 110 047 471 544 271 935 597 971 263 314 727 177 997 393 143 620 495 113 567 893 979 ⋱
232 771 067 239 978 005 987 338 346 262 198 732 407 404 789 003 414 529 832 126 794 080 403 332 940 022 230 919 621 105 760 ⋱
630 452 643 860 474 769 166 941 065

In[103]:=
```
DecodeRSA[result]
```

Out[103]= What about this message?

■ Finally a really secret message:

In[104]:=
```
<< "C:\\Dokumente und Einstellungen\\koepf\\Eigene
    Dateien\\Koepf\\Vorträge\\Kamerun2010\\message"
```

In[105]:=
```
result = EncodeRSA[message]
```

Out[105]= 4 439 117 834 308 799 218 567 451 267 793 619 968 652 806 352 082 537 022 318 047 476 119 156 657 755 612 963 294 931 118 379 ⋱
906 184 553 622 663 129 743 171 675 171 647 344 610 991 515 283 670 303 332 668 847 385 196 416 171 245 941 646 813 271 612 ⋱
160 958 678 502 167 240 728 547 512 736 283 292 036 935 985 802 172 434 947 564 693 448 867 039 793 223 559 619 583 123 701 ⋱
314 145 784 272 996 616 494 422 905 418 341 064 847 161 205 947 164 402 789 788 105 126 830 649 961 674 915 578 096 889 710 ⋱
140 748 827 570 685 309 751 724 030

In[106]:=
```
DecodeRSA[result]
```

Out[106]= This is my last message with RSA! The final topic of the talk is Coding Theory.

A two-correcting Reed-Solomon Code

- **These functions generate the necessary redundancy and are able to correct up to two errors.**

In[107]:= `code = ReedSolomon["MATHEMATICS", 31]`

Out[107]= \DPZMATHEMATICS

In[108]:= `InverseReedSolomon[code, 31]`

Out[108]= MATHEMATICS

In[109]:= `code1 = StringReplace[code, "H" -> "J"]`

Out[109]= \DPZMATJEMATICS

In[110]:= `InverseReedSolomon[code1, 31]`

Solve::svars : Equations may not give solutions for all "solve" variables. ≫

Out[110]= MATHEMATICS

In[111]:= `code2 = StringReplace[code1, "D" -> "Z"]`

Out[111]= \ZPZMATJEMATICS

In[112]:= `InverseReedSolomon[code2, 31]`

Out[112]= MATHEMATICS

In[113]:= `code3 = StringReplace[code2, "E" -> "F"]`

Out[113]= \ZPZMATJFMATICS

In[114]:= `InverseReedSolomon[code3, 31]`

Part::partw : Part −1 of {} does not exist. ≫

ReplaceAll::reps :
{{}[[−1]]} is neither a list of replacement rules nor a valid dispatch table, and so cannot be used for replacing. ≫

Part::partw : Part −1 of {} does not exist. ≫

ReplaceAll::reps :
{{}[[−1]]} is neither a list of replacement rules nor a valid dispatch table, and so cannot be used for replacing. ≫

Part::partw : Part −1 of {} does not exist. ≫

General::stop : Further output of Part::partw will be suppressed during this calculation. ≫

ReplaceAll::reps :
{{}[[−1]]} is neither a list of replacement rules nor a valid dispatch table, and so cannot be used for replacing. ≫

General::stop : Further output of ReplaceAll::reps will be suppressed during this calculation. ≫

Out[114]= MATJFMATICS