Threads Documents

Sorted Numbers

1 2 3 5 6 7 9 9 13 13 13 14 14 15 16 16 16 19 19 19 21 21 22 22 22 24 24 26 27 29 30 32 32 33 36 37 38 38 39 41 42 42 44 46 46 47 48 48 49 50 50 51 52 52 52 52 53 53 54 54 54 54 57 57 58 58 59 62 62 62 63 63 64 65 66 68 69 69 71 72 72 72 72 74 75 75 75 76 78 78 78 79 79 79 81 82 82 82 84 85 85 85 86 86 86 87 87 89 89 90 91 91 92 93 93 94 96 96 99 99 100 106 106 106 107 108 108 108 109 110 110 112 112 115 116 117 118 118 118 118 119 119 121 121 121 121 123 123 123 124 124 126 126 129 130 131 131 133 135 136 136 138 141 145 145 147 147 148 148 148 149 149 149 151 152 154 154 156 157 157 158 160 161 161 161 164 166 169 169 170 171 174 174 176 179 182 183 183 184 184 185 186 186 186 188 192 193 195 199 201 201 201 202 203 204 204 205 205 205 206 206 208 208 210 210 210 213 214 216 217 220 221 221 223 223 223 223 224 224 224 226 227 227 231 233 235 235 237 237 238 238 242 243 244 244 245 247 248 252 252 252 252 253 254 254 255 255 255 256 256 257 257 263 265 267 267 269 269 270 272 273 274 276 278 279 280 280 281 282 282 284 287 289 289 289 290 292 293 293 294 295 296 297 298 298 298 302 303 304 305 309 309 311 315 316 316 316 317 318 319 319 324 324 325 325 325 327 328 328 329 330 331 332 332 333 337 341 343 343 346 347 349 349 351 352 354 355 355 355 355 357 358 358 360 361 362 362 363 363 363 364 366 370 370 371 372 372 373 376 376 377 377 378 379 379 379 380 381 382 382 382 384 385 390 391 391 392 393 393 393 394 394 395 395 395 396 396 398 399 399 401 404 406 407 407 407 407 408 408 409 410 411 411 412 416 416 417 417 420 420 421 421 422 423 426 426 427 429 429 430 430 431 431 434 435 437 437 440 441 441 443 445 446 446 448 449 450 452 455 456 456 457 457 457 458 458 460 461 461 464 465 465 465 465 466 467 468 469 469 470 470 471 472 473 475 475 476 477 477 477 480 480 482 482 483 484 485 486 486 486 487 488 489 489 492 492 492 492 495 495 496 496 496 499 501 505 505 506 506 507 507 509 510 511 511 513 513 514 515 516 518 520 520 521 521 522 523 523 523 524 528 528 528 528 529 530 530 530 531 531 532 534 536 538 539 544 544 544 544 548 550 550 554 554 554 556 558 559 560 560 563 564 565 567 568 569 571 572 573 573 573 574 575 578 578 579 579 581 582 583 584 584 584 585 585 585 587 587 588 588 588 588 589 590 593 595 601 601 605 605 608 609 611 612 612 613 613 614 614 615 615 616 617 618 618 619 620 620 621 623 625 625 626 626 629 629 629 630 630 631 631 631 631 632 634 634 634 637 639 645 646 646 647 647 648 648 651 651 655 656 657 657 658 658 662 662 663 663 664 665 669 669 669 671 674 674 674 675 676 676 677 677 677 677 677 678 680 681 682 684 684 684 684 686 686 687 687 688 688 688 690 691 691 692 693 694 694 695 695 695 695 696 698 699 701 703 706 707 707 707 707 708 708 711 714 714 714 715 717 718 719 720 722 723 725 726 727 727 728 729 729 730 730 730 731 731 731 732 734 735 736 737 738 739 739 740 740 740 743 743 744 744 745 745 746 747 747 749 752 754 756 758 758 758 761 761 764 766 766 768 769 769 770 770 771 771 771 771 773 773 773 775 776 776 778 778 781 781 784 784 785 787 787 788 789 789 792 792 792 792 793 794 795 795 796 799 801 801 802 802 802 802 802 803 805 805 806 807 807 808 808 809 810 810 810 810 811 812 812 813 813 814 815 816 819 821 821 823 823 823 823 824 824 824 825 826 827 827 828 828 830 830 832 832 833 834 835 837 838 838 839 842 845 846 848 849 849 851 852 853 855 856 856 857 858 860 860 861 861 861 861 863 864 866 866 867 867 869 869 869 869 869 870 871 874 875 878 878 880 881 883 885 890 893 893 893 894 897 898 899 899 900 900 903 906 909 909 912 913 913 913 914 916 917 922 923 924 925 926 929 930 930 931 933 935 938 938 938 939 940 945 946 946 948 949 949 952 953 953 953 953 954 956 957 959 960 960 961 962 962 962 963 963 963 963 964 964 965 965 971 971 972 974 974 976 976 977 978 978 979 980 980 980 983 984 985 985 985 985 986 987 988 989 989 991 992 993 996 997 999

Copied Text

In the series of figures below, a sequence of passes is shown for the binary search. Let's go over them step-by-step. The first step is to look at the array in it's initial state. We are going to have to keep three "pointers" into the array for this algorithm - three integer variables that contain the indicies of three different places that we are concerned with in the array: the low index that we are still looking at, the high index that we are still looking at, and the midpoint index between the low and high. The figure below shows you these values. The low and high indices are the first and last element indices of the array, and the midpoint is shown to be (low+high)/2. Note that we need to do integer division to find the midpoint. That way, if the number of elements in the array is even, and thus the "midpoint" is actually not an element, we will set the mid pointer to one less than what floating point division would give us. If you didn't catch on to what integer division did way back at the beginning of the semester, now is the time to make sure you do. So if there are 8 elements, then (0+7)/2 would be 3.5 with floating point division, but will return 3 with integer division.

In the series of figures below, a sequence of passes is shown for the binary search. Let's go over them step-by-step. The first step is to look at the array in it's initial state. We are going to have to keep three "pointers" into the array for this algorithm - three integer variables that contain the indicies of three different places that we are concerned with in the array: the low index that we are still looking at, the high index that we are still looking at, and the midpoint index between the low and high. The figure below shows you these values. The low and high indices are the first and last element indices of the array, and the midpoint is shown to be (low+high)/2. Note that we need to do integer division to find the midpoint. That way, if the number of elements in the array is even, and thus the "midpoint" is actually not an element, we will set the mid pointer to one less than what floating point division would give us. If you didn't catch on to what integer division did way back at the beginning of the semester, now is the time to make sure you do. So if there are 8 elements, then (0+7)/2 would be 3.5 with floating point division, but will return 3 with integer division.

In the series of figures below, a sequence of passes is shown for the binary search. Let's go over them step-by-step. The first step is to look at the array in it's initial state. We are going to have to keep three "pointers" into the array for this algorithm - three integer variables that contain the indicies of three different places that we are concerned with

with in the array: the low index that we are still looking at, the high index that we are still looking at, and the midpoint index between the low and high. The figure below shows you these values. The low and high indices are the first and last element indices of the array, and the midpoint is shown to be (low+high)/2. Note that we need to do integer division to find the midpoint. That way, if the number of elements in the array is even, and thus the "midpoint" is actually not an element, we will set the mid pointer to one less than what floating point division would give us.
If you didn't catch on to what integer division did way back at the beginning of the semester, now is the time to make sure you do.
So if there are 8 elements, then (0+7)/2 would be 3.5 with floating point division, but will return 3 with integer division.
In the series of figures below, a sequence of passes is shown for the binary search. Let's go over them step-by-step. The first step is to look at the array in it's initial state. We are going to have to keep three "pointers" into the array for this algorithm - three integer variables that contain the indicies of three different places that we are concerned
with in the array: the low index that we are still looking at, the high index that we are still looking at, and the midpoint index between the low and high. The figure below shows you these values. The low and high indices are the first and last element indices of the array, and the midpoint is shown to be (low+high)/2. Note that we need to do integer division to find the midpoint. That way, if the number of elements in the array is even, and thus the "midpoint" is actually not an element, we will set the mid pointer to one less than what floating point division would give us.
If you didn't catch on to what integer division did way back at the beginning of the semester, now is the time to make sure you do.
So if there are 8 elements, then (0+7)/2 would be 3.5 with floating point division, but will return 3 with integer division.
In the series of figures below, a sequence of passes is shown for the binary search. Let's go over them step-by-step. The first step is to look at the array in it's initial state. We are going to have to keep three "pointers" into the array for this algorithm - three integer variables that contain the indicies of three different places that we are concerned with in the array: the low index that we are still looking at, the high index that we are still looking at, and the midpoint index between the low and high. The figure below shows you these values. The low and high indices are the first and last element indices of the array, and the midpoint is shown to be (low+high)/2. Note that we need to do integer division to find the midpoint. That way, if the number of elements in the array is even, and thus the "midpoint" is actually not an element, we will set the mid pointer to one less than what floating point division would give us.

Fibonnacci sequence

0

1

1

2

3

5

8

13

21

34

55

89

144

233

377

610

987

1597

2584

4181

6765

10946

17711

28657

46368

75025

121393

196418

317811

514229

832040

1346269

2178309

3524578

5702887

9227465

14930352

24157817

39088169

63245986

102334155