

## Source Code :

```
#include <stdlib.h>
#include<stdio.h>
int array[1000];
int num=1000,i;
void main()
{
    int inputs=0,ch;
    time_t t;
    printf("The following %d random numbers have been assigned\n\n",num);
    srand(time(&t));
    for(inputs=0;inputs<num;inputs++)
    {
        array[inputs]=rand()%1000;
        printf("%d ",array[inputs]);
    }
    printf("\n\n");
    printf("\n1.BUBBLE SORT\n2.INSERTION SORT\n3.SELECTION SORT\n4.MERGE
SORT\n5.QUICK SORT\n6.EXIT");
    printf("\nPLEASE ENTER YOUR CHOICE\n");
    scanf("%d", & ch);
    switch (ch)
    {
    case 1:
        bubbleSort();
        break;
    case 2:
        insertionSort();
        break;
    case 3:
        selectionSort();
        break;
    case 4:
        {mergeSort(0,num-1);
        printf("\n\nYOU HAVE SELECTED MERGE SORT\nAfter sorting:\n");
        for(i=0; i<num; i++)
            printf("%d ", array[i]);
        printf("\n");
        break;}
    case 5:
        quickSort(0,num-1);
        printf("\n\nYOU HAVE SELECTED QUICK SORT\nAfter sorting:\n");
        for(i=0; i<num; i++)
            printf("%d ", array[i]);
        printf("\n");
        break;
    case 6:
        printf("YOU HAVE SELECTED TO EXIT");
        exit(0);
        break;
    }
}
```

```

bubbleSort()
{
    int count, dim, swap;
    printf("\nYOU HAVE SELECTED BUBBLE SORT\n");
    for (count=0;count<(num-1);count++)
    {
        for (dim=0;dim<num-count-1;dim++)
        {
            if (array[dim] > array[dim+1])
            {
                swap = array[dim];
                array[dim] = array[dim+1];
                array[dim+1] = swap;
            }
        }
    }
    printf("\nSorted List is:\n");
    for (count=0;count<num;count++)
    {
        printf("%d ", array[count]);
    }
}

insertionSort()
{
    int count, dim, temp;
    printf("\nYOU HAVE SELECTED INSERTION SORT\n");
    for (count=1;count<= num-1;count++)
    {
        dim = count;
        while (dim>0 && array[dim]<array[dim-1])
        {
            temp = array[dim];
            array[dim] = array[dim-1];
            array[dim-1] = temp;
            dim--;
        }
    }
    printf("\nSorted List is:\n");
    for (count=0;count<=num-1;count++)
    {
        printf("%d ", array[count]);
    }
}

selectionSort()
{
    int count, dim, pos, swap;
    printf("\nYOU HAVE SELECTED SELECTION SORT\n");
    for (count=0;count<(num-1);count++)
    {
        pos = count;
        for (dim=count+1;dim<num;dim++)
        {
            if ( array[pos] > array[dim] )

```

```

        {
            pos = dim;
        }
    }
    if(pos!=count)//For swapping
    {
        swap = array[count];
        array[count] = array[pos];
        array[pos] = swap;
    }
}
printf("\nSorted list is:\n");
for (count=0;count<num;count++)
{
    printf("%d ", array[count]);
}
}
mergeSort(int beg, int end)
{
    int mid;
    if (beg < end)
    {
        mid = (beg+end)/2;
        mergeSort( beg, mid);
        mergeSort( mid+1, end);
        merge_array(beg, mid, end);
    }
}
void merge_array(int beg, int mid, int end)
{
    int i, left_end, num, temp, j, k, b[50];
    for(i=beg; i<=end; i++)
    b[i] = array[i];
    i = beg;
    j = mid+1;
    k = beg;
    while ((i<=mid) && (j<=end))
    {
        if (b[i] <= b[j])
        {
            array[k] = b[i];
            i++;
            k++;
        }
        else
        {
            array[k] = b[j];
            j++;
            k++;
        }
    }
    if (i <= mid)
    {

```

```

        while (i <= mid)
        {
            array[k] = b[i];
            i++;
            k++;
        }
    }
    else
    {
        while (j <= end)
        {
            array[k] = b[j];
            j++;
            k++;
        }
    }
}

quickSort(int beg, int end)
{
    int x;
    if (beg < end)
    {
        x = partition( beg, end);
        quickSort( beg, x-1);
        quickSort( x+1, end);
    }
}

int partition( int beg, int end)
{
    int loc = beg, temp;
    while (1)
    {
        while (array[loc]<=array[end] && loc!=end)
            end--;
        if (loc == end)
            return loc;
        temp = array[loc];
        array[loc] = array[end];
        array[end] = temp;
        loc = end;
        while (array[loc]>=array[beg] && loc!=beg)
            beg++;
        if (loc == beg)
            return loc;
        temp = array[loc];
        array[loc] = array[beg];
        array[beg] = temp;
        loc = beg;
    }
}

```

## Sample Input :

```
"E:\T4--ashesh_sort all types.exe"

The following 1000 random numbers have been assigned

296 983 491 825 220 617 150 277 946 942 699 325 218 606 417 159 394 975 940 135
538 47 151 271 667 361 400 925 715 619 968 341 844 421 555 745 871 84 485 814 55
4 607 414 57 122 10 528 682 653 321 246 120 462 983 823 708 230 281 155 66 809 3
64 405 990 792 134 162 395 5 299 193 834 916 950 203 865 983 219 661 282 466 26
707 709 265 119 768 408 375 176 483 494 321 551 356 111 738 49 256 527 488 790 4
1 688 166 449 745 828 987 250 496 965 699 386 397 504 341 79 841 995 117 745 578
522 43 836 278 832 883 63 110 587 166 972 974 41 42 857 50 722 229 935 962 286
583 665 481 569 457 838 539 867 144 491 958 853 143 763 962 129 724 40 715 721 5
69 151 575 331 342 894 75 252 21 761 787 410 182 862 82 358 436 726 534 274 672
885 863 191 845 227 913 59 891 652 625 464 744 213 533 312 4 784 110 48 659 570
642 747 392 482 413 257 248 725 601 239 285 672 558 633 517 245 971 684 778 3 18
1 817 883 935 15 815 490 255 699 441 655 552 608 16 942 533 645 113 844 172 693
634 631 28 99 433 511 490 919 151 448 700 681 707 697 213 526 217 402 2 874 916
665 556 405 785 390 223 272 935 160 986 169 340 959 169 333 313 627 680 414 145
493 856 517 454 44 294 978 437 867 994 761 659 402 246 391 399 897 568 46 658 44
745 262 316 347 47 597 689 881 727 519 668 104 319 801 611 67 548 458 939 917 6
31 443 245 720 709 569 745 455 783 977 146 328 375 379 67 476 349 250 832 42 593
707 263 75 209 971 888 55 461 158 268 879 657 558 482 460 851 113 24 889 54 630
191 596 694 33 639 323 500 213 782 20 892 321 555 29 78 958 159 46 410 984 511
915 944 539 571 911 462 72 746 853 371 63 282 576 332 215 162 998 428 893 203 71
2 130 882 175 886 749 944 992 761 355 405 68 784 939 537 407 343 455 901 969 837
391 124 66 980 113 718 76 911 797 723 688 767 857 549 371 238 320 441 882 672 1
44 376 883 79 291 32 814 478 862 232 597 695 345 685 120 273 911 278 827 836 7 6
94 292 744 49 123 399 901 853 915 21 931 976 183 213 990 920 994 313 250 835 803
4 581 899 873 482 293 252 574 945 615 961 520 463 418 804 44 596 132 613 923 99
3 907 173 522 857 6 946 413 47 277 877 756 952 246 219 608 586 938 18 855 37 919
69 238 882 783 800 927 73 713 671 516 79 182 709 34 927 751 529 73 160 693 846
411 976 274 746 835 560 246 207 312 720 20 937 378 83 582 607 29 23 29 69 763 22
790 609 231 616 145 797 783 421 548 20 193 621 616 352 754 458 243 940 10 457 7
90 446 492 605 702 375 245 948 296 581 995 958 745 321 728 355 709 845 839 61 10
5 264 238 582 522 717 356 266 167 456 957 488 396 211 923 373 785 449 880 130 56
3 489 98 184 513 876 854 961 561 578 864 240 612 26 17 27 839 721 890 895 708 11
7 32 134 376 122 272 245 918 452 109 161 476 271 216 995 773 196 808 488 655 564
970 278 533 177 178 204 247 515 395 186 537 865 219 721 97 497 931 772 405 434
53 943 285 983 518 543 884 551 199 74 237 179 295 424 244 175 779 674 26 220 936
906 237 69 751 428 224 697 547 667 766 793 936 462 765 347 196 212 728 102 506
167 461 998 996 389 873 594 745 308 521 692 64 287 914 681 892 740 91 50 24 105
201 50 389 942 827 353 465 779 328 593 386 525 6 103 28 46 764 899 183 755 674 2
91 660 186 351 237 921 155 27 388 363 422 431 309 988 989 985 731 798 483 996 42
3 135 918 152 76 597 657 941 906 299 742 30 649 92 922 548 465 919 16 496 279 74
8 114 86 98 647 453 510 479 226 957 420 578 456 729 546 238 935 488 413 783 781
541 429 597 983 743 721 908 909 443 563 794 690 374 15 529 451 261 346 30 38 296
808 928 693 583 134 645 67 427 402 585 118 736 80 473 286 57 766 999 324 637 81
8 742 958 857 50 833 899 654 733 623 10 756 816 466 64 487 459 88 505 825 813 48
9 278 224 789 347 672 737 99 754 700 573 759 899 112 75 704 363 357 647 975 700
13 57 317 768 726 689 897 865 88 194 535 256 332 676 894 564 374 46 281 613 52 3
07 176 81 199 773 471 823 137 872 512 203 871 776 93 298 714 749 192 774 609 974
670 67 993 76 865 667 879 712 390 64 429 440 927 304 59 622 297 573 888 433 80
581 737 488 985 175 466 308

1.BUBBLE SORT
2.INSERTION SORT
3.SELECTION SORT
4.MERGE SORT
```

## Sample Output :

```
"E:\T4--ashesh_sort all types.exe"
PLEASE ENTER YOUR CHOICE
3
YOU HAVE SELECTED SELECTION SORT
Sorted list is:
2 3 4 4 5 6 6 7 10 10 10 13 15 15 16 16 17 18 20 20 20 21 21 22 23 24 24 26 26 2
6 27 27 28 28 29 29 29 30 30 32 32 33 34 37 38 40 41 41 42 42 43 44 44 44 46 46
46 46 47 47 47 48 49 49 50 50 50 50 52 53 54 55 57 57 57 59 59 61 63 63 64 64 64
66 66 67 67 67 67 68 69 69 69 72 73 73 74 75 75 75 76 76 76 78 79 79 79 80 80 8
1 82 83 84 86 88 88 91 92 93 97 98 98 99 99 102 103 104 105 105 109 110 110 111
112 113 113 113 114 117 117 118 119 120 120 122 122 123 124 129 130 130 132 134
134 134 135 135 137 143 144 144 145 145 146 150 151 151 151 152 155 155 158 159
159 160 160 161 162 162 166 166 167 167 169 169 172 173 175 175 175 176 176 177
178 179 181 182 182 183 183 184 186 186 191 191 192 193 193 194 196 196 199 199
201 203 203 203 204 207 209 211 212 213 213 213 215 216 217 218 219 219 219
220 220 223 224 224 226 227 229 230 231 232 237 237 237 238 238 238 239 240
243 244 245 245 245 245 246 246 246 246 247 248 250 250 250 252 252 255 256 256
257 261 262 263 264 265 266 268 271 271 272 272 273 274 274 277 277 278 278 278
278 279 281 281 282 282 285 285 286 286 287 291 291 292 293 294 295 296 296 296
297 298 299 299 304 307 308 308 309 312 312 313 313 316 317 319 320 321 321 321
321 323 324 325 328 328 331 332 332 333 340 341 341 342 343 345 346 347 347 347
349 351 352 353 355 355 356 356 357 358 361 363 363 364 371 371 373 374 374 375
375 375 376 376 378 379 386 386 388 389 389 390 390 391 391 392 394 395 395 396
397 399 399 400 402 402 402 405 405 405 405 407 408 410 410 411 413 413 413 414
414 417 418 420 421 421 422 423 424 427 428 428 429 429 431 433 433 434 436 437
440 441 441 443 443 446 448 449 449 451 452 453 454 455 455 456 456 457 457 458
458 459 460 461 461 462 462 462 463 464 465 465 466 466 466 471 473 476 476 478
479 481 482 482 482 483 483 485 487 488 488 488 488 489 489 490 490 491 491
492 493 494 496 496 497 500 504 505 506 510 511 511 512 513 515 516 517 517 518
519 520 521 522 522 522 525 526 527 528 529 529 533 533 533 534 535 537 537 538
539 539 541 543 546 547 548 548 549 551 551 552 554 555 555 556 558 558 560
561 563 563 564 564 568 569 569 569 570 571 573 573 574 575 576 578 578 581
581 581 582 582 583 583 585 586 587 593 593 594 596 596 597 597 597 597 601 605
606 607 607 608 608 609 609 611 612 613 613 615 616 616 617 619 621 622 623 625
627 630 631 631 633 634 637 639 642 645 645 647 647 649 652 653 654 655 655 657
657 658 659 659 660 661 665 665 667 667 667 668 670 671 672 672 672 672 674 674
676 680 681 681 682 684 685 688 688 689 689 690 692 693 693 693 694 694 695 697
697 699 699 699 700 700 700 702 704 707 707 707 708 708 709 709 709 712 712
713 714 715 715 717 718 720 720 721 721 721 721 722 723 724 725 726 726 727 728
728 729 731 733 736 737 737 738 740 742 742 743 744 744 745 745 745 745 745
745 746 746 747 748 749 749 751 751 754 754 755 756 756 759 761 761 761 763 763
764 765 766 766 767 768 768 772 773 773 774 776 778 779 779 781 782 783 783 783
783 784 784 785 785 787 789 790 790 790 792 793 794 797 797 798 800 801 803 804
808 808 809 813 814 814 815 816 817 818 823 823 825 825 827 827 828 832 832 833
834 835 835 836 836 837 838 839 839 841 844 844 845 845 846 851 853 853 853 854
855 856 857 857 857 857 862 862 863 864 865 865 865 866 867 867 871 871 872 873
873 874 876 877 879 879 880 881 882 882 882 883 883 884 885 886 888 888 889
890 891 892 892 893 894 894 895 897 897 899 899 899 899 901 901 906 906 907 908
909 911 911 911 913 914 915 915 916 916 917 918 918 919 919 919 920 921 922 923
923 925 927 927 927 928 931 931 935 935 935 935 936 936 937 938 939 939 940 940
941 942 942 942 943 944 944 945 946 946 948 950 952 957 957 958 958 958 959
961 961 962 962 965 968 969 970 971 971 972 974 974 975 975 976 976 977 978 980
983 983 983 983 983 984 985 985 986 987 988 989 990 990 992 993 993 994 994 995
995 995 996 996 998 998 999
Process returned 1000 (0x3E8)    execution time : 101.928 s
Press any key to continue.
```