

DEPARTMENT OF CYBER SECURITY



DESING AND ANALYSIS OF ALGORITHM ASSINGMENT

SUBMITTED BY

211042 Noman Masood Khan A

SEMESTER

Fifth

SUBJECT

Design and analysis of algorithm

SUBMITTED TO

Dr. Ammar Masood

a. Explanation for the Insertion Sort and Merge Sort Code

i. Insertion Sorting Code

Programming language:

C++

```
#include<iostream>
using namespace std;

int main(){

    int A[10]={6,4,8,9,1,5,7,2,3,10},key,j;

    for (int i = 0; i<10; i++){

        key = A[i];

        j = i-1;

        while (j >= 0 && A[j]>key){

            A[j+1] = A[j];
            j=j-1;

        }
        A[j+1] = key;
    }

    for (int j = 0; j<10; j++){
        cout<<A[j]<<"\n";
    }

}
```

The program is basically working on the user initialized array.

The working principal of the program is that it is working on the key and the element of an array and their comparison to sort the element.

For loop is being used to iterate over all the elements of the array. In each iteration, A[i] element is being taken as a key and the element behind that key is being taken as j.

While the j does not reach the 0 and we found out the greater element than the key, till then we will quit the loop else the loop will continue.

In the while loop condition, we are checking that if the element at the j = i-1 position is greater than the key, that means that it is already sorted and quit the loop.

In the body of the while loop, when we have found that element less than the key, then put this element in the sorted array. At the j+1 position.

a. Merge Sorting code

The merge sorting code is working on the divide and conquer technique which is being done on the recursion.

The program has three parts:

- i. The main function
- ii. mergeSort function
- iii. merge function

i. Main function

```
int main() {  
    int arr[] = {6, 5, 12, 10, 9, 1};  
    int size = sizeof(arr) / sizeof(arr[0]);  
    mergeSort(arr, 0, size - 1);  
    cout << "Sorted array: \n";  
    for (int i = 0; i < size; i++)  
        cout << arr[i] << " "<<endl;  
}
```

The main function has the initialized array and it is calculating the size of the given array.

Then it is calling the mergeSort function will run the recursion and to divide the arrays to the pair to twos.

At-last the sorted array will be displayed to the user.

ii. MergeSort Function

```
void mergeSort(int arr[], int l, int r) {  
    if (l < r) {  
        // m is the point where the array is divided  
        int m = l + (r - l) / 2;  
  
        mergeSort(arr, l, m);  
        mergeSort(arr, m + 1, r);  
  
        // Merge the sorted subarrays  
        merge(arr, l, m, r);  
    }  
}
```

This function is taking the array and the index of the right and left sub-arrays.

Then mergeSort function is starting the recursion to break the sub-arrays to the sub-arrays and passing the indexes of the sub-arrays to the itself function.

First the left array is being sent to the mergeSort function, which it is dividing it to the more sub-parts of left and right arrays.

Then the mergeSort function is being run on the right sub-array to divide it to the sub-arrays.

When the both of them have been divided then the code will call the merge function to merge the left and right arrays.

iii. Merge function

```

void merge(int arr[], int p, int q, int r) {
    int n1 = q - p + 1, n2 = r - q;
    int L[n1], M[n2], i=0, j=0, k=p;

    for (int i = 0; i < n1; i++)
        L[i] = arr[p + i];

    for (int j = 0; j < n2; j++)
        M[j] = arr[q + 1 + j];

    while (i < n1 && j < n2) {
        if (L[i] <= M[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = M[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = M[j];
        j++;
        k++;
    }
}

```

The function is making a new array for the left sub array and right sub arrays and filling the elements with the for loops.

Then while loop is being run to compare the element of the newly created arrays to sort them in the in the arr array. The comparison is being done on the newly created left and right arrays L and M. Then the sorted element is being found and that is being put to the arr array. The arr array will be sorted array which will be merge and sorted.

The elements left out in any L or M array will instead to the arr array because the left out elements will be the one which were left out because the loop ended.

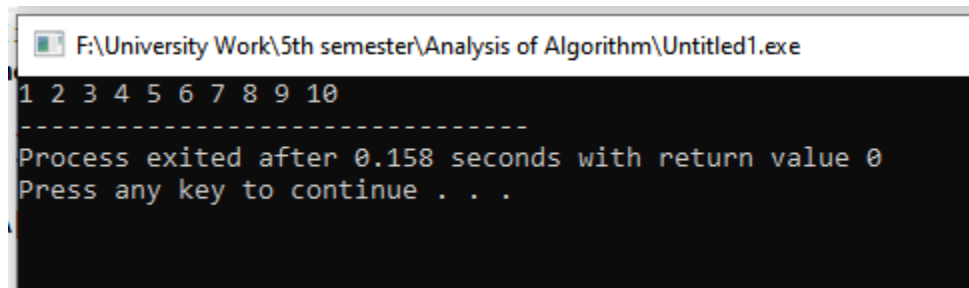
b. Test cases used to check the code

In the test cases I have provided the input to the code by initializing the array with the int numbers.

i. First test case for the insertion sorting

Input : {6,4,8,9,1,5,7,2,3,10}

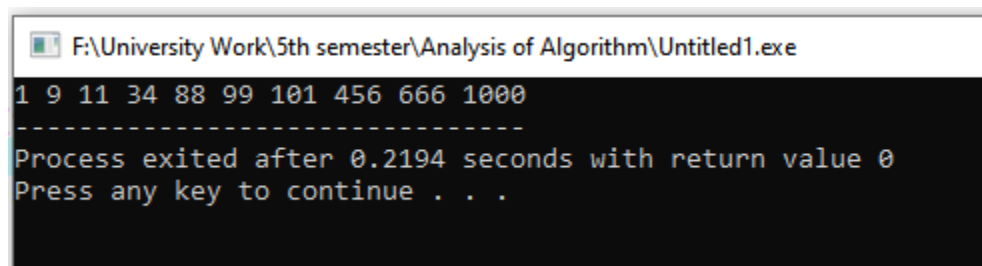
Output :



```
F:\University Work\5th semester\Analysis of Algorithm\Untitled1.exe
1 2 3 4 5 6 7 8 9 10
-----
Process exited after 0.158 seconds with return value 0
Press any key to continue . . .
```

Input: int A[10]={9,101,88,1,34,11,99,1000,666,456}

Output :



```
F:\University Work\5th semester\Analysis of Algorithm\Untitled1.exe
1 9 11 34 88 99 101 456 666 1000
-----
Process exited after 0.2194 seconds with return value 0
Press any key to continue . . .
```

ii. Insertion Sorting

Input: `int arr[] = {6, 5, 12, 10, 9, 1, 7, 2, 13, 20}`

Output:

```
F:\University Work\5th semester\Analysis of Algorithm\Untitled2.exe
Sorted array:
1 2 5 6 7 9 10 12 13 20
-----
Process exited after 0.2325 seconds with return value 0
Press any key to continue . . .
```

Input :

`int arr[] = {200,300,100,400,900,502,444,302,101,999}`

Output:

```
F:\University Work\5th semester\Analysis of Algorithm\Untitled2.exe
Sorted array:
100 101 200 300 302 400 444 502 900 999
-----
Process exited after 0.2237 seconds with return value 0
Press any key to continue . . .
```

c. Procedure for generation of inputs

The method for the input generation is the random input generator function

I am using the random generator in the code to generate the 10 elements and store them inside the array. Here the size of the input being generated is 10 and the numbers are less than the 100.

The code for the random input generator is :

```
int length=10;

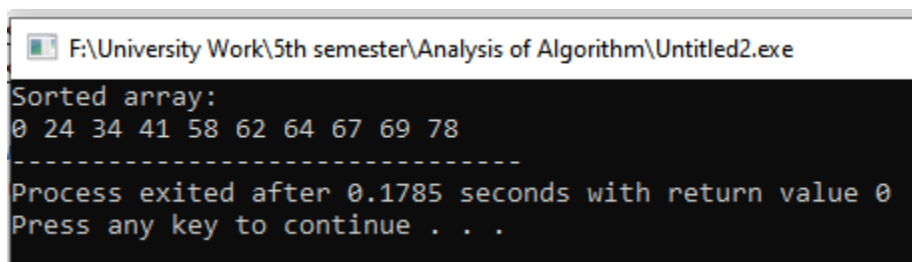
int arr[length];

for (int i=0;i<length;i++){

    arr[i] = rand() % 100;

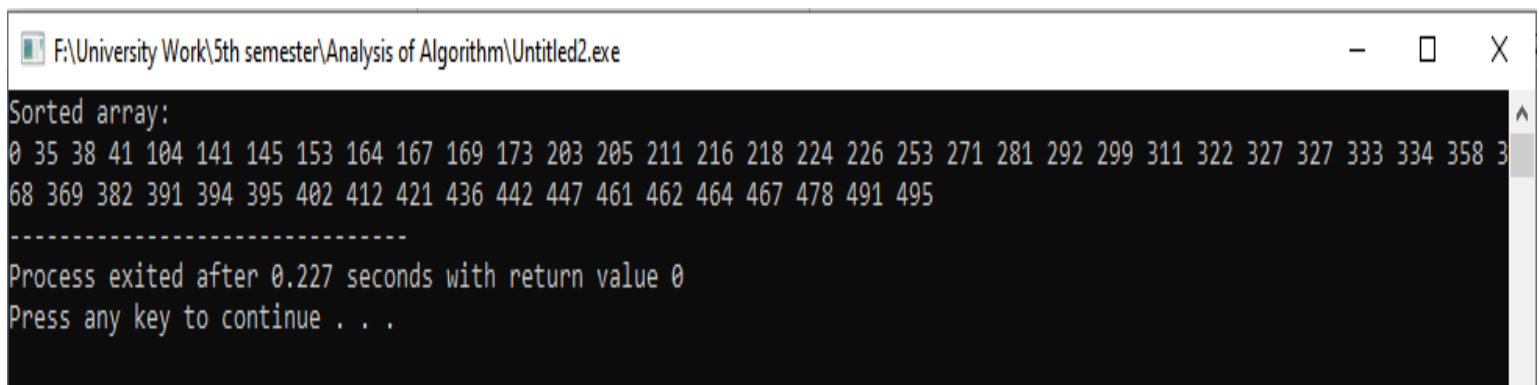
}
```

The output of the merge sorting for the 10 random numbers less than 100 being sorted is



```
F:\University Work\5th semester\Analysis of Algorithm\Untitled2.exe
Sorted array:
0 24 34 41 58 62 64 67 69 78
-----
Process exited after 0.1785 seconds with return value 0
Press any key to continue . . .
```

Now I am going to test the program on the 50 inputs which are less than 500, then the all the numbers where sorted perfectly



```
F:\University Work\5th semester\Analysis of Algorithm\Untitled2.exe
Sorted array:
0 35 38 41 104 141 145 153 164 167 169 173 203 205 211 216 218 224 226 253 271 281 292 299 311 322 327 327 333 334 358 368 369 382 391 394 395 402 412 421 436 442 447 461 462 464 467 478 491 495
-----
Process exited after 0.227 seconds with return value 0
Press any key to continue . . .
```


For the **insertion sorting** I am also doing the same and giving the **70 inputs** to the code which will be numbers between 0 to 1000. So the **merge sort result** before and after sorting can be seen is :

```

F:\University Work\5th semester\Analysis of Algorithm\Untitled1.exe
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491 995 942 827 436 391 604 902 153 292 382 421 716 718 895 4
47 726 771 538 869 912 667 299 35 894 703 811 322 333 673 664 141 711 253 868 547 644 662 757 37 859 723 741 529 778 316
35 190 842 288 106 40 942 264 648

-----After Sorting-----
35 35 37 40 41 106 141 145 153 169 190 253 264 281 288 292 299 316 322 333 334 358 382 391 421 436 447 464 467 478 491 5
00 529 538 547 604 644 648 662 664 667 673 703 705 711 716 718 723 724 726 741 757 771 778 811 827 827 842 859 868 869 8
94 895 902 912 942 942 961 962 995

-----
Process exited after 0.2846 seconds with return value 0
Press any key to continue . . .

```

I have checked both the algorithms with the input size of 1000 and both the algorithms were performing very well as they were sorting the random inputs into the correct order.

```

-----After Sorting-----
1 2 3 3 3 6 7 7 8 8 8 8 9 10 10 11 11 15 15 17 18 18 19 20 21 21 21 22 22 23 24 25 27 28 28 28 30 30 31 31 35 35 35 36 37 37 38 38 39 40 40 40 41 41 41 43 44 44 49 50 5
2 53 53 53 55 58 58 60 60 60 61 64 67 67 70 71 71 72 72 72 72 73 75 75 75 75 77 80 80 82 82 84 84 85 86 87 87 88 88 90 93 93 97 98 99 101 102 103 105 106 107 108 109 10
9 110 111 112 112 113 113 114 115 115 116 116 117 118 119 123 124 125 127 129 129 129 131 132 132 139 140 141 141 142 142 142 142 144 145 145 145 146 148 150 152 15
2 153 153 153 154 154 155 156 157 159 161 161 164 164 168 168 168 169 169 169 170 170 171 173 173 174 175 176 177 179 181 181 182 184 185 185 186 186 187 188 189 190 19
0 191 191 192 192 193 193 195 195 195 196 196 199 200 200 200 202 202 202 205 205 209 212 213 213 213 215 215 216 220 221 221 221 221 222 222 223 224 224 226 227 22
9 230 232 233 234 235 240 240 245 247 249 249 253 253 255 256 256 257 258 259 260 261 262 262 263 264 264 264 264 264 269 270 270 270 271 272 279 279 281 281 281 282 28
2 285 285 286 286 287 287 288 288 289 290 291 292 292 292 292 295 296 296 297 297 299 300 302 303 303 303 303 303 306 308 309 309 310 313 313 313 313 313 314 314 314 31
5 316 316 317 318 318 318 321 322 322 323 323 324 326 328 329 330 333 333 334 334 335 337 337 342 342 343 347 348 350 350 350 350 353 353 355 355 355 355 356 357 357 35
7 358 359 359 360 361 361 362 363 363 365 368 369 369 370 371 372 374 375 376 380 382 383 384 386 389 391 391 392 392 393 398 401 402 404 405 409 410 410 411 411 413 41
3 413 414 416 416 416 418 421 421 422 422 423 423 423 423 424 425 426 426 428 428 429 430 430 432 433 434 436 437 439 439 439 441 443 446 447 448 450 451 452 454 45
5 457 457 458 459 460 461 462 463 464 464 466 466 467 467 467 472 472 474 474 476 477 477 477 478 480 481 482 483 483 483 484 484 485 485 487 487 488 488 489 489 49
1 492 493 496 497 498 500 503 503 503 504 505 506 508 508 510 510 511 511 512 512 514 515 515 518 519 519 520 520 523 525 526 527 527 529 529 532 534 535 536 537 537 53
8 538 538 539 540 540 541 541 541 542 543 543 545 546 547 547 548 548 548 549 549 549 549 550 555 555 555 556 556 556 556 557 558 559 561 561 565 565 565 565 570 573 574 57
6 576 577 578 580 584 584 584 585 585 587 588 588 589 589 589 591 591 593 593 595 596 596 598 600 600 600 601 601 601 602 604 604 605 606 607 608 608 609 610 611 616 61
7 617 617 618 619 619 620 622 623 624 624 625 625 625 626 626 626 626 627 627 627 629 629 629 629 634 634 634 635 636 637 637 637 639 641 643 644 646 646 648 648 64
8 649 650 650 651 651 652 653 654 655 655 657 658 658 659 662 662 662 663 664 667 667 668 668 670 671 673 673 673 674 675 676 676 678 678 678 679 681 683 685 685 686 68
7 688 689 690 690 692 693 694 694 695 695 696 698 699 700 701 701 702 703 704 704 704 705 705 705 706 710 711 711 712 712 713 716 717 718 718 721 722 723 723 724 724 72
5 726 726 728 729 734 734 734 734 736 737 740 741 741 745 745 748 748 750 752 753 753 754 756 756 757 757 757 758 758 759 759 760 760 762 763 763 763 766 767 769 77
1 771 773 774 775 777 778 778 781 783 783 786 786 787 788 788 789 790 790 796 798 798 798 800 801 802 805 807 808 811 812 813 813 814 815 815 815 818 818 823 824 824 82
5 825 827 827 829 829 829 831 831 832 832 833 833 835 836 838 840 841 842 843 844 844 847 848 850 850 851 851 853 855 855 855 858 859 861 864 865 866 867 868 869 86
9 869 869 869 870 870 874 875 875 877 878 881 881 882 885 886 887 888 888 890 892 893 893 894 895 896 896 898 900 900 900 900 901 902 902 902 903 905 909 909 911 912 91
2 913 913 913 923 923 924 924 924 926 928 929 930 931 932 932 932 934 935 936 937 938 938 938 940 941 941 942 942 943 944 944 944 945 945 945 946 948 948 949 949 951 95
4 954 954 955 956 958 958 958 959 961 961 962 962 962 962 963 964 966 966 969 970 971 971 971 972 972 974 974 975 976 977 977 982 985 985 986 989 990 992 993 993 994 99
5 996 997 998 999

```

d. Comparison of the time and memory complexity

Time Complexity for Insertion Sorting:

Best Case:

The best running time is $O(n)$

Average Case:

The average running time is $O(n^2)$.

Worst Case:

The Best case running time is $O(n^2)$.

Time complexity for the merge sorting

Best Case:

The best running time is $O(n \log n)$

Average Case:

The average running time is $O(n \log n)$.

Worst Case:

The Best case running time is $O(n \log n)$.

Memory Complexity for insertion sorting

Memory or space complexity for the insertion sorting is $O(1)$. This is because the space required by the insertion sorting is constant and does not depend on the size of an input.

Memory Complexity for merge sorting

Memory or space complexity for the merge sorting is $O(n)$. Merge sort being recursive takes up the additional space to store the elements in the memory.

Conclusion

I compared the result time of both the algorithms by giving both the input size of the 5000 (five thousand). The merge sort gave back the result in very short time but the insertion sorting took time. This is because of the reason that the merge sorting takes $n \log n$ and the insertion sort takes n^2 time complexity. With the inputs growing to bigger and bigger, the time to sort is growing for the insertion sort while it does not affect merge sort pretty much. Merge sort takes memory but it does not affect the efficiency of the merge sort algorithm.