CS 202 HW1

Section: 1

Name: Barkın Saday

ID: 21902967

Question 1)

a)

Show $f(n) = 8n^4 + 5n^3 + 7$ is $O(n^5)$:

- $T(n) = O(f(n))$ if there are positive constants c and n_0 such that $T(n) <= cf(n)$ when $n > n_0$

If we use the values $n_0 = 9$, $c = 1$ for any $n >= 9$ than n^5 is an upper bound of f(n).

b)

[22, 8, 49, 25, 18, 30, 20, 15, 35, 27]

Selection Sort:

- Let's use two indices to trace the algorithm, called: **curr** and **large**

- We will also use an index n which will be the outer loop

curr: Goes from the beginning of the array to the end of it one by one (iteratively)

large: Index of the largest value so far

-We will compare the elements at the indices and if array[curr] > array[large] large will be equal to curr

- After curr reaches the last element we will put the element in **large** at the end of the array (move data) which is the **sorted** part of the array.

-Then we will continue by resetting the indices (curr = 0, large = 0) and apply the same procedure. However, curr will go up to "size-**n**"**,** not "size-1" (the last element will not be included). So, basically, we will keep working on the **unsorted** part of the array and the **sorted** part is the end of the array up to n many items (n is the number of iterations done so far).

First iteration (n=1):   (n will go up to size-1)

Step 1:

large = 0 -> 22 (assign)

curr = 0 -> 22 (assign)

22 > 22 (false)

Step 2:

large = 0 -> 22

curr = 1 -> 8 (assign)

8 >22 (false)

Step 3:

large = 0 -> 22

curr = 2 -> 49 (assign)

49 > 22 (true) -> large = 2 ->49 (assign)

Step 4:

large = 2 ->49

curr = 3 -> 25 (assign)

25 > 49 (false)

Step 5:

large = 2 -> 49

curr = 4 -> 18

18 > 49 (false)

Step 6:

large = 2 -> 49

curr = 5 -> 30

30 > 49 (false)

Step 7:

large = 2 -> 49

curr = 6 -> 20

20 > 49 (false)

Step 8:

large = 2 -> 49

curr = 7 -> 15

15 > 49 (false)

Step 9:

large = 2 -> 49

curr = 8 -> 35

35 > 49 (false)

Step 10:

large = 2 -> 49

curr = 9 -> 27

27 > 49 (false)

Step 11 (move data):

swap(n-1, large); (move data)

array-> [22, 8, 27, 25, 18, 30, 20, 15, 35, 49]

continue with n = 2 ...

Bubble Sort:

Once again we will work with array as if it has two parts **sorted** and **unsorted**

We will use two indices **i** and **j**

j: will go through the sorted part until j+1 reaches the last element of the sorted part

i: will go from 0 to size-n

We will compare j with j+1 and swap them if j > j+1 then continue by increasing j. After j+1 reaches the end of the sorted part we will update the sorted part (increase n) and restart the procedure

Step 1_1:

j = 0 -> 22

j+1 = 1 -> 8

22 > 8 (true) -> swap(j, j+1)   (move data)

array -> [8, 22, 49, 25, 18, 30, 20, 15, 35, 27]

Step 1_2:

j = 1 -> 22

j+1 = 2 -> 49

22 > 49 (false)

array -> [8, 22, 49, 25, 18, 30, 20, 15, 35, 27]

Step 1_3:

j = 2 -> 49

j+1 = 3 -> 25

49 > 25 (true) - > swap(j+ j+1)   (move data)

array -> [8, 22, 25, 49, 18, 30, 20, 15, 35, 27]

Step 1_4:

j = 3 -> 49

j+1 = 4 -> 18

49 > 18 (true) -> swap(j, j+1)   (move data)

array -> [8, 22, 25, 18, 49, 30, 20, 15, 35, 27]

Step 1_5:

j = 4 -> 49

j+1 = 5 -> 30

49 > 30 (true) - > swap(j+ j+1)   (move data)

array -> [8, 22, 25, 18, 30, 49, 20, 15, 35, 27]

Step 1_6:

j = 5 -> 49

j+1 = 6 -> 20

49 > 20 (true) - > swap(j+ j+1)   (move data)

array -> [8, 22, 25, 18, 30, 20, 49, 15, 35, 27]

Step 1_6:

j = 6 -> 49

j+1 = 7 -> 15

49 > 15 (true) - > swap(j+ j+1)   (move data)

array -> [8, 22, 25, 18, 30, 20, 15, 49, 35, 27]

Step 1_6:

j = 7 -> 49

j+1 8-> 35

49 > 35 (true) - > swap(j+ j+1)   (move data)

array -> [8, 22, 25, 18, 30, 20, 15, 35, 49, 27]

Step 1_7:

j = 8 -> 49

j+1 9 -> 27

49 > 27 (true) - > swap(j+ j+1)   (move data)

array -> [8, 22, 25, 18, 30, 20, 15, 35, 27, 49]

Continue with step 2_1 (n = 2) we will go up to size-n ({49} is now on **sorted** part)

. . . . .

Question 2)

a)

- "main.cpp", "sorting.cpp", "sorting.h" are in the zip file

b)

- "main.cpp" is in the zip file

c)

-"makefile" is in the zip file

Screenshot for part C:

```
C:\Users\asus>ssh barkin.saday@dijkstra.ug.bcc.bilkent.edu.tr
barkin.saday@dijkstra.ug.bcc.bilkent.edu.tr's password:
Last login: Tue Mar  8 04:51:29 2022 from 139.179.211.58
[barkin.saday@dijkstra ~]$ ls
Cabinet.cpp  Flower.h          FlowerList.cpp  hw1_202          main.cpp       myProgramHW2    sorting.cpp
Cabinet.h    FlowerLibrary.cpp FlowerList.h    LabOrganizer.cpp makefile       myProgramHW3    sorting.h
Flower.cpp   FlowerLibrary.h   hw1             LabOrganizer.h   myProgramHW1   myProgramHW3v2
[barkin.saday@dijkstra ~]$ make
make: `hw1_202' is up to date.
[barkin.saday@dijkstra ~]$ ./hw1_202
Insertion Sort:
{9, 6, 7, 16, 18, 5, 2, 12, 20, 1, 16, 17, 4, 11, 13, 8}
{1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 16, 16, 17, 18, 20}
Comparison: 58, Movement: 88
============================================================================
Bubble Sort:
{9, 6, 7, 16, 18, 5, 2, 12, 20, 1, 16, 17, 4, 11, 13, 8}
{1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 16, 16, 17, 18, 20}
Comparison: 58, Movement: 174
============================================================================
Merge Sort:
{9, 6, 7, 16, 18, 5, 2, 12, 20, 1, 16, 17, 4, 11, 13, 8}
{1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 16, 16, 17, 18, 20}
Comparison: 47, Movement: 128
============================================================================
Quick Sort:
{9, 6, 7, 16, 18, 5, 2, 12, 20, 1, 16, 17, 4, 11, 13, 8}
{1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 16, 16, 17, 18, 20}
Comparison: 27, Movement: 114
[barkin.saday@dijkstra ~]$
```

d)

Performance Analysis:

Random Array:

```
Performance Analysis for Random Array / Size: 5000
==============================================================================
Insertion Sort:
Execution time for Insertion Sort: 31 ms
Comparison: 6120734 Movement: 6130732
==============================================================================
Bubble Sort:
Execution time for Bubble Sort: 78 ms
Comparison: 6120734 Movement: 18362202
==============================================================================
Merge Sort:
Execution time for Merge Sort: 0 ms
Comparison: 55128 Movement: 123616
==============================================================================
Quick Sort:
Execution time for Quick Sort: 0 ms
Comparison: 18079 Movement: 68937

Process returned 0 (0x0)   execution time : 0.203 s
Press any key to continue.
```

```
Performance Analysis for Random Array / Size: 10000
==============================================================================
Insertion Sort:
Execution time for Insertion Sort: 91 ms
Comparison: 24935268 Movement: 24955266
==============================================================================
Bubble Sort:
Execution time for Bubble Sort: 272 ms
Comparison: 24935268 Movement: 74805804
==============================================================================
Merge Sort:
Execution time for Merge Sort: 16 ms
Comparison: 120140 Movement: 267232
==============================================================================
Quick Sort:
Execution time for Quick Sort: 16 ms
Comparison: 40495 Movement: 151185

Process returned 0 (0x0)   execution time : 0.470 s
Press any key to continue.
```

```
C:\Dersler\CS202\Homeworks\HW1\CS202_HW1\bin\Debug\CS202_HW1.exe
Comparison: 47, Movement: 128
============================================================================
Quick Sort:
{9, 6, 7, 16, 18, 5, 2, 12, 20, 1, 16, 17, 4, 11, 13, 8}
{1, 2, 4, 5, 6, 7, 8, 9, 11, 12, 13, 16, 16, 17, 18, 20}
Comparison: 27, Movement: 114



Performance Analysis for Random Array / Size: 15000
=============================================================================
Insertion Sort:
Execution time for Insertion Sort: 201 ms
Comparison: 56266361 Movement: 56296359
=============================================================================
Bubble Sort:
Execution time for Bubble Sort: 612 ms
Comparison: 56266361 Movement: 168799083
=============================================================================
Merge Sort:
Execution time for Merge Sort: 0 ms
Comparison: 188914 Movement: 417232
=============================================================================
Quick Sort:
Execution time for Quick Sort: 0 ms
Comparison: 51801 Movement: 200103

Process returned 0 (0x0)   execution time : 0.926 s
Press any key to continue.
```

```
Performance Analysis for Random Array / Size: 20000
=============================================================================
Insertion Sort:
Execution time for Insertion Sort: 335 ms
Comparison: 99019635 Movement: 99059633
=============================================================================
Bubble Sort:
Execution time for Bubble Sort: 1076 ms
Comparison: 99019635 Movement: 297058905
=============================================================================
Merge Sort:
Execution time for Merge Sort: 16 ms
Comparison: 260282 Movement: 574464
=============================================================================
Quick Sort:
Execution time for Quick Sort: 16 ms
Comparison: 82926 Movement: 308478

Process returned 0 (0x0)   execution time : 1.521 s
Press any key to continue.
```

```
Performance Analysis for Random Array / Size: 25000
==========================================================================
Insertion Sort:
Execution time for Insertion Sort: 549 ms
Comparison: 154695862 Movement: 154745860
==========================================================================
Bubble Sort:
Execution time for Bubble Sort: 1741 ms
Comparison: 154695862 Movement: 464087586
==========================================================================
Merge Sort:
Execution time for Merge Sort: 0 ms
Comparison: 333302 Movement: 734464
==========================================================================
Quick Sort:
Execution time for Quick Sort: 0 ms
Comparison: 97589 Movement: 367467

Process returned 0 (0x0)   execution time : 2.368 s
Press any key to continue.
```

```
Performance Analysis for Random Array / Size: 30000
====================================================================
Insertion Sort:
Execution time for Insertion Sort: 753 ms
Comparison: 222594132 Movement: 222654130
====================================================================
Bubble Sort:
Execution time for Bubble Sort: 2500 ms
Comparison: 222594132 Movement: 667782396
====================================================================
Merge Sort:
Execution time for Merge Sort: 0 ms
Comparison: 407698 Movement: 894464
====================================================================
Quick Sort:
Execution time for Quick Sort: 0 ms
Comparison: 136686 Movement: 499758

Process returned 0 (0x0)   execution time : 3.367 s
Press any key to continue.
```

```
Performance Analysis for Random Array / Size: 35000
============================================================================
Insertion Sort:
Execution time for Insertion Sort: 1003 ms
Comparison: 303515736 Movement: 303585734
============================================================================
Bubble Sort:
Execution time for Bubble Sort: 3441 ms
Comparison: 303515736 Movement: 910547208
============================================================================
Merge Sort:
Execution time for Merge Sort: 16 ms
Comparison: 483371 Movement: 1058928
============================================================================
Quick Sort:
Execution time for Quick Sort: 16 ms
Comparison: 121425 Movement: 468975

Process returned 0 (0x0)   execution time : 4.554 s
Press any key to continue.
```

```
Performance Analysis for Random Array / Size: 40000
============================================================================
Insertion Sort:
Execution time for Insertion Sort: 1311 ms
Comparison: 396656780 Movement: 396736778
============================================================================
Bubble Sort:
Execution time for Bubble Sort: 4546 ms
Comparison: 396656780 Movement: 1189970340
============================================================================
Merge Sort:
Execution time for Merge Sort: 0 ms
Comparison: 560450 Movement: 1228928
============================================================================
Quick Sort:
Execution time for Quick Sort: 0 ms
Comparison: 149333 Movement: 567699

Process returned 0 (0x0)   execution time : 5.967 s
Press any key to continue.
```

Question 3)

-In this experiment, I used 4 different algorithms for sorting integer arrays with different sizes and different distributions. For small-sized arrays, I was not expecting much difference in terms of elapsed time for sorting the arrays, which was a correct assumption since for smaller inputs elapsed time was very close to zero. However, for larger inputs, I noticed that merge sort and quick sort were working faster than insertion sort and bubble sort since insertion sort and bubble sort have time complexity $O(n^2)$. Also, I keep track of the number of key comparisons and data movement for each algorithm. I noticed that for very small-sized inputs (like an array of 20 or 30 items) number of comparisons of insertion sort and bubble sort were usually smaller than quick sort and merge sort. However, for much larger inputs number of comparisons was increasing significantly for the bubble sort and insertion sort algorithms. The same thing applies to data movement as well. Also, the bubble sort algorithm seems to have the most number of data movements and the insertion sort algorithm has the most number of key comparisons. Insertion sort and bubble sort were not that much affected by the type of the array (almost sorted / almost unsorted). However, the quicksort was observably working slower if the array was almost sorted.