

Project 1: Sorting Algorithms

2218-CSE-5311-002

Name: Sudharsan Rajam

Student ID: 1001874246

Description:

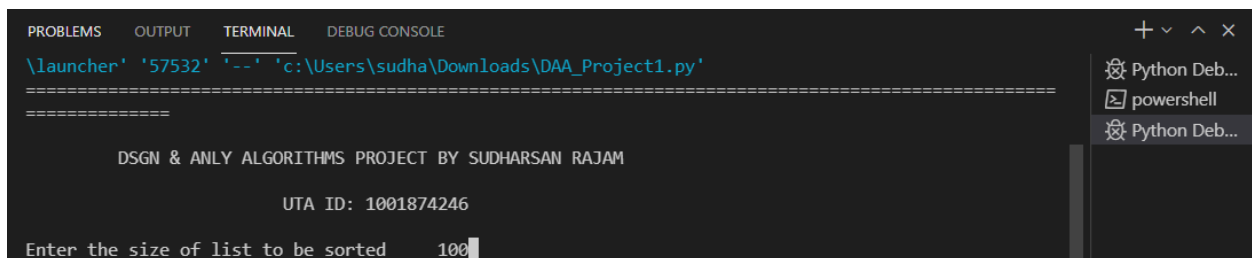
A sorting algorithm is an algorithm that puts elements of a list in a certain order. Efficient sorting is important for optimizing the efficiency of other algorithms (such as search and merge algorithms) that require input data to be in sorted lists. Sorting is the process of arranging data in an order, that order can be ascending order or descending order. In this project, I am sorting the data using 6 algorithms (Merge Sort, Heap Sort, Quick Sort (last element and using 3 medians), Insertion Sort, Selection Sort and Bubble Sort). First, I'm taking the input from the user about how many elements they want to sort after that, I am giving the option to the user about which sorting algorithm they want to use to sort the data. Moreover, I've given the users an option to choose where all of the algorithm will run on same data, resulting to which runtime and sorted array for all the algorithms is displayed. I have used Python as my coding language and Visual Studio Code as IDE to develop this program. For Input, I am taking random numbers from -5000 to 25000 using the random library.

User Interface:

The user interface of the project is as follows:

end, depending on the user's choice the sorted array and the runtime of that algorithm is displayed. If, user gives a wrong input he/she will be asked to give a valid input.

1) Size of the list (Let's say 100)



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
\launcher' '57532' '--' 'c:\Users\sudha\Downloads\DAA_Project1.py'
=====
DSGN & ANLY ALGORITHMS PROJECT BY SUDHARSAN RAJAM
UTA ID: 1001874246
Enter the size of list to be sorted 100
```

2) IP values are selected at random using randomInput function and print the values for user to see. And furthermore gives user the option to choose from 8 options.

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Enter the size of list to be sorted 100

-1585 1084 15829 -1614 10947 20790 -4034 24896 20092 -599 20294 5273 10613 23561 -2520 22242 6194 14
955 -2555 7754 -4003 -3346 -1258 12456 22015 14118 -2209 10644 -3166 20125 1477 -3001 18913 16328 -4
906 24289 -1807 23360 11045 20896 24832 5201 22893 11726 7286 7710 24955 13586 -735 8944 5150 16669
14651 17494 23696 18375 21052 17122 17048 2099 11358 13133 3001 -964 6937 23645 17627 6624 291 1223
731 3502 -3413 5261 12156 1800 21444 17791 5770 16131 5436 17928 6008 9393 16703 24768 19674 10843 2
2889 20385 -841 9284 21699 16868 9581 553 9362 9143 4668 15794

1. MERGE SORT
2. HEAP SORT
3. REGULAR QUICK SORT
4. QUICK SORT USING 3 MEDIANS
5. INSERTION SORT
6. SELECTION SORT
7. BUBBLE SORT
8. COMPARE ALL OF THE ABOVE ALGORITHMS AND THEIR RUNNING TIME
```

3) Finally depending upon user choice it'll show you the sorted list and the time it executed in.
(Here I have selected option 8 which chooses all of the above algos)

```
CHOOSE FROM 1 - 8 TO PERFORM THE SORTING ALGOS
8
Sorted elements for merge sort :
-4700 -4195 -4025 -3611 -3246 -2575 -2468 -2433 -2366 -2344 -2321 -2237 -2000 -1715 -1550 -1393 -1222 -640 -428 -205 -66 703 1056 1125 1399 1819 1935 2419 2
475 2832 2870 2911 3113 3335 3480 3758 3918 4094 4349 4935 5363 5850 6204 6520 6768 6802 7181 7244 7334 7431 7735 8622 9055 9119 9312 10220 10799 11134 1135
3 11987 12008 12026 12165 12432 12677 12816 12821 13220 14171 14842 15240 15554 16002 16096 16297 16401 16425 17903 18145 19423 20546 20725 20772 20844 2087
9 20910 20981 20990 21213 21289 21427 22078 22120 22245 22306 22989 23362 23399 23475 23843
Sorted elements for heap sort :
-4700 -4195 -4025 -3611 -3246 -2575 -2468 -2433 -2366 -2344 -2321 -2237 -2000 -1715 -1550 -1393 -1222 -640 -428 -205 -66 703 1056 1125 1399 1819 1935 2419 2
475 2832 2870 2911 3113 3335 3480 3758 3918 4094 4349 4935 5363 5850 6204 6520 6768 6802 7181 7244 7334 7431 7735 8622 9055 9119 9312 10220 10799 11134 1135
3 11987 12008 12026 12165 12432 12677 12816 12821 13220 14171 14842 15240 15554 16002 16096 16297 16401 16425 17903 18145 19423 20546 20725 20772 20844 2087
9 20910 20981 20990 21213 21289 21427 22078 22120 22245 22306 22989 23362 23399 23475 23843
Sorted elements for quick sort :
-4700 -4195 -4025 -3611 -3246 -2575 -2468 -2433 -2366 -2344 -2321 -2237 -2000 -1715 -1550 -1393 -1222 -640 -428 -205 -66 703 1056 1125 1399 1819 1935 2419 2
475 2832 2870 2911 3113 3335 3480 3758 3918 4094 4349 4935 5363 5850 6204 6520 6768 6802 7181 7244 7334 7431 7735 8622 9055 9119 9312 10220 10799 11134 1135
3 11987 12008 12026 12165 12432 12677 12816 12821 13220 14171 14842 15240 15554 16002 16096 16297 16401 16425 17903 18145 19423 20546 20725 20772 20844 2087
9 20910 20981 20990 21213 21289 21427 22078 22120 22245 22306 22989 23362 23399 23475 23843
Sorted elements for quick sort using 3 medians sort :
-4700 -4195 -4025 -3611 -3246 -2575 -2468 -2433 -2366 -2344 -2321 -2237 -2000 -1715 -1550 -1393 -1222 -640 -428 -205 -66 703 1056 1125 1399 1819 1935 2419 2
475 2832 2870 2911 3113 3335 3480 3758 3918 4094 4349 4935 5363 5850 6204 6520 6768 6802 7181 7244 7334 7431 7735 8622 9055 9119 9312 10220 10799 11134 1135
3 11987 12008 12026 12165 12432 12677 12816 12821 13220 14171 14842 15240 15554 16002 16096 16297 16401 16425 17903 18145 19423 20546 20725 20772 20844 2087
9 20910 20981 20990 21213 21289 21427 22078 22120 22245 22306 22989 23362 23399 23475 23843
Sorted elements for insertion sort:
-4700 -4195 -4025 -3611 -3246 -2575 -2468 -2433 -2366 -2344 -2321 -2237 -2000 -1715 -1550 -1393 -1222 -640 -428 -205 -66 703 1056 1125 1399 1819 1935 2419 2
475 2832 2870 2911 3113 3335 3480 3758 3918 4094 4349 4935 5363 5850 6204 6520 6768 6802 7181 7244 7334 7431 7735 8622 9055 9119 9312 10220 10799 11134 1135
3 11987 12008 12026 12165 12432 12677 12816 12821 13220 14171 14842 15240 15554 16002 16096 16297 16401 16425 17903 18145 19423 20546 20725 20772 20844 2087
9 20910 20981 20990 21213 21289 21427 22078 22120 22245 22306 22989 23362 23399 23475 23843
Sorted elements for selection sort :
-4700 -4195 -4025 -3611 -3246 -2575 -2468 -2433 -2366 -2344 -2321 -2237 -2000 -1715 -1550 -1393 -1222 -640 -428 -205 -66 703 1056 1125 1399 1819 1935 2419 2
475 2832 2870 2911 3113 3335 3480 3758 3918 4094 4349 4935 5363 5850 6204 6520 6768 6802 7181 7244 7334 7431 7735 8622 9055 9119 9312 10220 10799 11134 1135
3 11987 12008 12026 12165 12432 12677 12816 12821 13220 14171 14842 15240 15554 16002 16096 16297 16401 16425 17903 18145 19423 20546 20725 20772 20844 2087
9 20910 20981 20990 21213 21289 21427 22078 22120 22245 22306 22989 23362 23399 23475 23843
```

```
Sorted elements for bubble sort:
-4700 -4195 -4025 -3611 -3246 -2575 -2468 -2433 -2366 -2344 -2321 -2237 -2000 -1715 -1550 -1393 -1222 -640 -428 -205 -66 703 1056 1125 1399 1819 1935 2419 2
475 2832 2870 2911 3113 3335 3480 3758 3918 4094 4349 4935 5363 5850 6204 6520 6768 6802 7181 7244 7334 7431 7735 8622 9055 9119 9312 10220 10799 11134 1135
3 11987 12008 12026 12165 12432 12677 12816 12821 13220 14171 14842 15240 15554 16002 16096 16297 16401 16425 17903 18145 19423 20546 20725 20772 20844 2087
9 20910 20981 20990 21213 21289 21427 22078 22120 22245 22306 22989 23362 23399 23475 23843

Merge Sort Algorithm Time          0.0008544921875
Heap Sort Algorithm Time           0.001032114028930664
Quick Sort Algorithm Time          0.0009617805480957031
Quick Sort Algorithm with 3 Median Time 0.0019941329956054688
Insertion Sort Algorithm Time       0.0009987354278564453
Selection Sort Algorithm Time       0.0010182857513427734
Bubble Sort Algorithm Time          0.001974821090698242
```

Merge Sort:

Merge sort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sub list consists of a single element and merging those sub lists in a manner that results into a sorted list. In my code, I have created a function for merge sort named “MergeSort” which takes the list as an input and returns the sorted array, and I’ve called this function from the main function(“MainSort”). The runtime complexities of merge sort in best, average and worst case are: $O(n \log n)$ and the worst space complexity is: $O(n)$.

Heap Sort:

Heap is a special case of balanced complete binary tree data structure where the root-node key is compared with its children and arranged accordingly. Generally, Heaps can be of two types Max Heap and Min Heap from which we pop the root of the tree and place it at the last of the array. I have defined a separate function for heap sort named “HeapSort” which takes list as an input and returns the sorted array. I am calling this HeapSort from the same main function(“MainSort”). The runtime complexities of heap sort in best, average and worst case are: $O(n \log n)$ and the worst space complexity is: $O(1)$.

Quick Sort using last element as the pivot:

Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways for this program, I have chosen last element as pivot. I have defined a separate function for quick sort named “RegularQuickSort” which takes list as an input and returns the sorted array. I am calling this RegularQuickSort from the same main function(“MainSort”). The runtime complexities of Quick sort in best, average and worst case are: $O(n \log n)$, $O(n \log n)$ and $O(n^2)$ and the worst space complexity is: $O(n)$.

Quick Sort using 3 medians:

The best case for quick sort is that if we could find the middle element. It is difficult to find the median so consider another strategy where we Choose the median of the first,

middle, and last entries in the list. This will usually give a better approximation of the actual median. It is done to avoid the worst case of quick sort in which time complexity is $O(n^2)$. In the code, I have defined a separate function for quick sort using 3 medians named "SpecialQuickSort" which takes list as an input and returns the sorted array. I am calling this SpecialQuickSort from the same main function("MainSort"). The runtime complexities of heat sort in best, average and worst case are: $O(n \log n)$ and the worst space complexity is: $O(n)$.

Insertion Sort:

Insertion sort is based on the idea that one element from the input elements is consumed in each iteration to find its correct position. In the code, I have defined a separate function for insertion sort named "InsertionSort" which takes list as an input and returns the sorted array. I am calling this InsertionSort from the same main function("MainSort"). The runtime complexities of Insetion sort in best, average and worst case are: $O(n)$, $O(n^2)$ and $O(n^2)$ and the worst space complexity is: $O(1)$.

Selection Sort:

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts the subarray which is already sorted and remaining array which is unsorted. Here, I have created a separate function for selection sort named "SelectionSort" in the code which takes list as an input and returns the sorted array. I am calling this SelectionSort from the same main function("MainSort"). The runtime complexities of Selection sort in best, average and worst case are: $O(n^2)$ and the worst space complexity is: $O(1)$.

Bubble Sort:

Bubble sort is the easiest sorting Algorithm that works by repeatedly swapping the adjacent elements if they are in unsorted order. Bubble sort compares each element with its adjacent number and swaps if it is smaller and It passes through the whole list n times until the whole list is sorted. I have defined a separate function for bubble sort named "BubbleSort" in the code which takes list as an input and returns the sorted array. I am calling this BubbleSort from the same main function("MainSort"). The runtime complexities of heat sort in best, average and worst case are: $O(n)$, $O(n^2)$ and $O(n^2)$ and the worst space complexity is: $O(1)$.

Comparison of all the sorting algorithms with a small input (500 elements):

Time taken (seconds) to complete the sorting for small input was least for quick sort (0.002997159957885742 secs) and highest for bubble sort (0.03685331344604492secs). Insertion sort (0.0039882659912109375 secs) was the second best after quick sort.

```
Merge Sort Algorithm Time      0.004935741424560547
Heap Sort Algorithm Time       0.004986763000488281
Quick Sort Algorithm Time      0.002997159957885742
Quick Sort Algorithm with 3 Median Time 0.004983186721801758
Insertion Sort Algorithm Time   0.0039882659912109375
Selection Sort Algorithm Time   0.027973651885986328
Bubble Sort Algorithm Time      0.03685331344604492
```

Figure 1

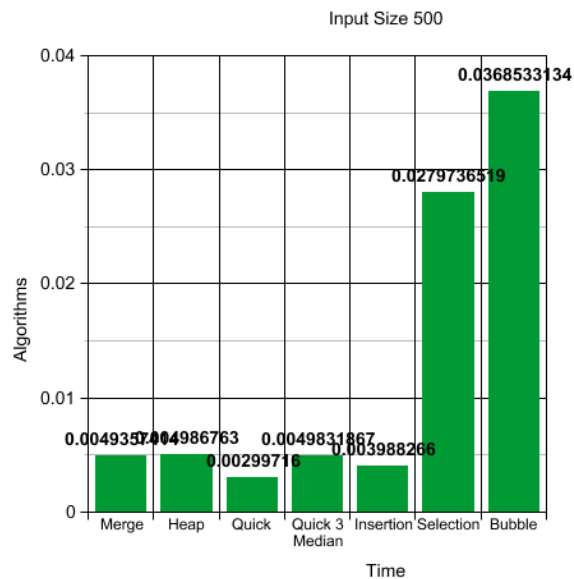


Figure 2

Comparison of all the sorting algorithms with a medium input (5000 elements):

Time taken (seconds) to complete the sorting for medium input was least for quick sort (0.06321907043457031 secs) and highest for Selection sort (2.3067569732666016 secs) by just a little bit of margin.

Merge Sort Algorithm Time	0.06478261947631836
Heap Sort Algorithm Time	0.12240719795227051
Quick Sort Algorithm Time	0.06321907043457031
Quick Sort Algorithm with 3 Median Time	0.07065987586975098
Insertion Sort Algorithm Time	0.5696041584014893
Selection Sort Algorithm Time	2.3067569732666016
Bubble Sort Algorithm Time	2.179497718811035

Figure 3

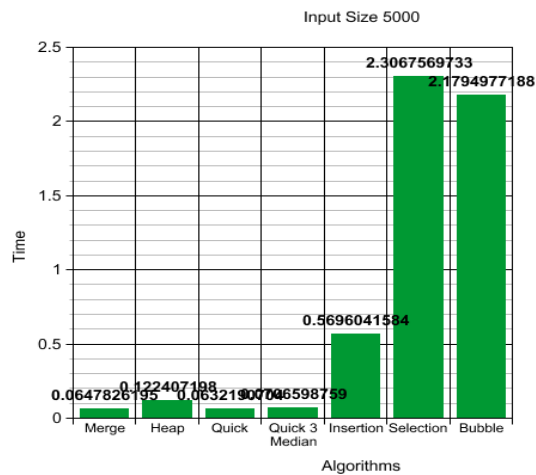


Figure 4

Comparison of all the sorting algorithms with a large input (15000 elements):

Time taken (seconds) to complete the sorting for large input was least for Merge sort (0.37010884284973145 secs) and highest for bubble sort (36.27246809005737 secs).

Merge Sort Algorithm Time	0.37010884284973145
Heap Sort Algorithm Time	0.6709296703338623
Quick Sort Algorithm Time	2.2515451908111572
Quick Sort Algorithm with 3 Median Time	1.6939737796783447
Insertion Sort Algorithm Time	1.627091884613037
Selection Sort Algorithm Time	20.765756607055664
Bubble Sort Algorithm Time	36.27246809005737

Figure 5

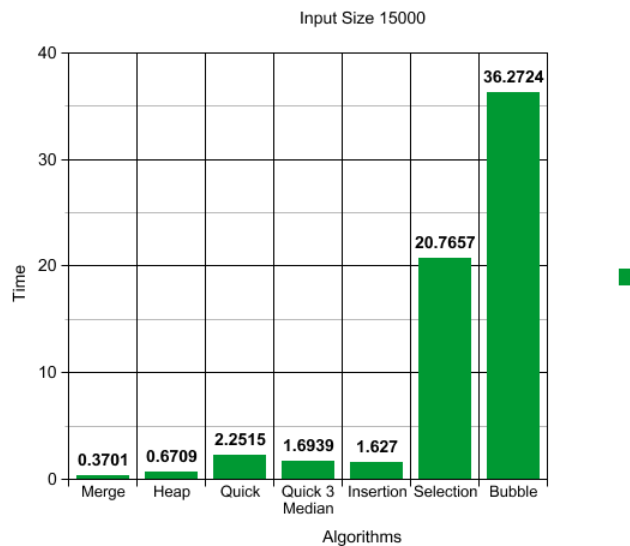


Figure 6

Conclusion

The time taken to sort for Bubble Sort was maximum in all three cases (small, medium and large input size). If we consider the average of all three cases quick sort has the best performance.