Sorting an array

Nariman Haidar

Fall term 2022

1 Introduction

In this assignment we are going to learn about sorting an fix size array. We are going to write a code to selection, insertion and merge algorithm and we are going to explain the run time as a function of the size of the array for each of those algorithms.

2 Method

Selection Sort:

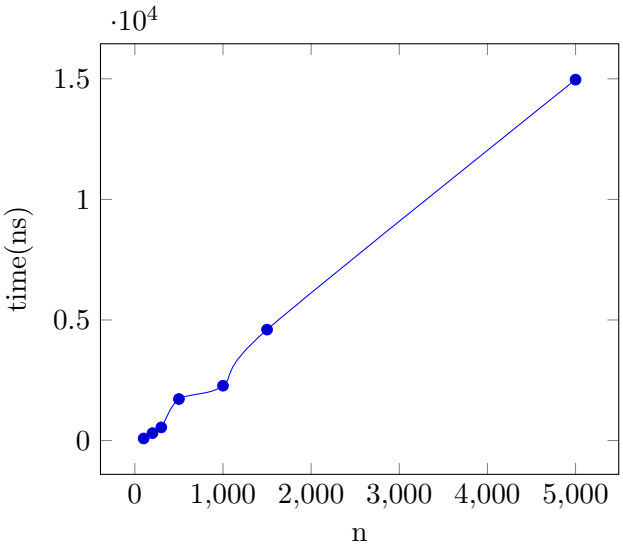
the selection sort algorithm sorts an array by first find the minimum element from the unsorted part and putting it at the beginning and it always keep to the ascending order and make the same thing every iteration. In our case first I started by I write a function that take an array like parameter and then I write a for loop that is going through all the elements in the array and then another for loop that going to compare between the elements if the first index of array is bigger than the other indexes in same array so we switch between those two index. Then I wrote a bench mark to get the time it took to sort en random array and we can see that in the table and in the graph.

Time complexity to this algorithm is o(n2) and that is because we have two loops: One loop to select an element of array O(n) and another loop is to compare that element with every other element O(n) so it is o(n2). The best Case Complexity is O(n2) It occurs when the array is already sorted. The worst Case Complexity is O(n2) and that because if the array is in ascending order and we want the array is in descending order.

code:

```
for (int i = 0; i < array.length-1; i++)
        {   int cand = i;
            for (int j = i+1; j < array.length; j++)
                if (array[j] < array[cand])
```

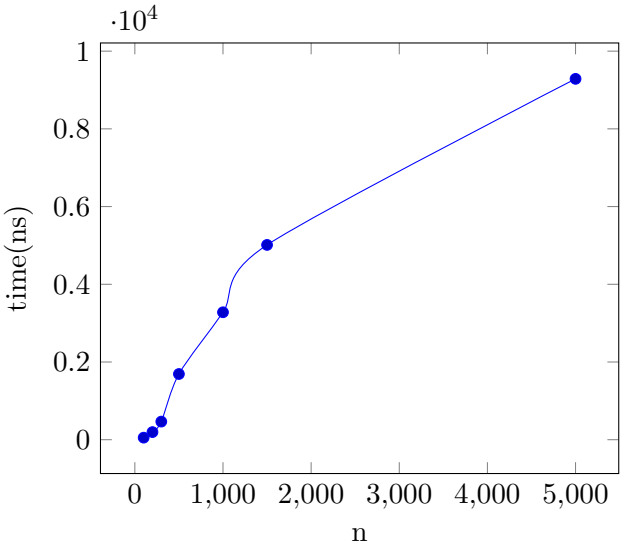
```
                cand = j;
            int k = array[cand];
            array[cand] = array[i];
            array[i] = k;
        }
```



Insertion Sort:

In the insertion sort we are going to see the first two elements of the array and then compare them if the first element is grater than the second element so we switch those two elements and then we going to the next element and look if the previous element is bigger than the new element so switch those and then we going to look if the new element that we sorted smaller than the element before so switch again until get the correct position and at the end finally, the array is completely sorted. Then I wrote a bench mark to get the time it took to sort en random array and we can see that in the table and in the graph.

Time complexity to the insertion algorithm is o(n2) because we are in every insertion swap the two elements. The worst Case Complexity: O(n2)and that because if the array is in ascending order and we want the array is in descending order. The best Case Complexity: O(n) and that is when the array is already sorted. So, there are only n number of comparisons.



Merge Sort:

The Merge Sort algorithm is an algorithm that called Divide and Conquer.

In merge sort we can see that the array is recursively divided into two halves until the size becomes 1, then merge processes comes into action and starts merging arrays back while sorting. First it starts with divided the original array ex: 3,2,5,11,23,7,1,12 to two arrays ex 3,2,5,11 23,7,1,12and then it will call itself to divide the first array that we get 3,2 5,11 and then it will call itself again 3 2 then it will start the merge with sort then 2,3 then 5 11 then merge them 5,11 then 2,3,5,11 then divide the second array 23,7 1,12 then 23 7 then merge 7,23 then divide 1 12 then merge 1,12 then merge1,7,12,23 then merge 1,2,3,5,7,11,12,23.

The complexity to the merge sort is o(nlogn) in all cases (worst, average, and best) because it always divides the array into two halves and takes linear time to merge two halves.

To implement the code first I started to write a method that is sort function that we will store temporary results in it before the final sorted array. Then I made a method that to divide the original array to half and then make it a gain and again until it just one element and that with use the recursion that it call itself. Then we are going to compare the first two elements in the first half array and see witch is smaller then it will put it first and do it with another two elements in the same half until it finishing the first half and then do the same thing with the another part of array and then when the both parts of array is sorted then merge the two arrays into one sorted array then with help with the merge method that I wrote merge part it will go through the two arrays item by item and select the smallest item found to be the next item in the list. The merge method first I made a for loop that it copy all items from the first index to the last index from the

original array to the copy array and then I made the merge with help av for loop for all indices from first index to last index and then I use if Sat's if i is greater than mid, move the j item to the original array and then update j. Then else if j is greater than last index, move the i item to the original array and then update i and then else if the i item is smaller than the j item so move it to the original array and then update i. Then I made some bench mark to measure the time that the random array takes to be sorted and we can see that in the table and in the graph.

code:

```
private static void sort(int[] org, int[] aux, int lo, int hi) {
       if (lo != hi) {
           int mid = lo + (hi - lo) / 2;
           // sort the items from lo to mid
           sort(org, aux, lo, mid);
           // sort the items from mid+1 to hi
           sort(org, aux, mid + 1, hi);
           // merge the two sections using the additional array
          merge(org, aux, lo, mid, hi);
           }
```

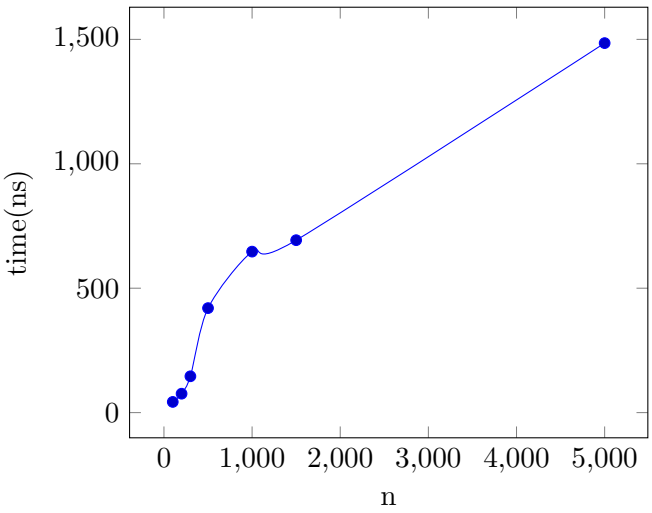


table:

As we see there is no so big difference between insertion sort and selection sort because they are both the time complexity is o(n2) so they take approximately the same time. In the merge sort it took less time than the insertion and selection sort because the complexity time is o(nlogn).

table

n	selection	insertion	merge
100	67	59	43
200	207	194	76
300	496	466	146
500	1721	1691	420
1000	2274	3280	647
1500	4600	5013	653

Table 1: This is the time out put of selection, insertion and merge to sorted an array