ITP Assignment 1

Sorting arrays into ascending or descending order, and printing the odd and even elements separately.

IEC2021063 Javesh Lodha, IEC2021064 Yash Gupta,
IEC2021065 Shashi Kumar Chaubey,
IEC2021066 Pavitra Pandey
Indian Institute of Information Technology, Allahabad
23-01-2022

Abstract: In this paper we have devised an algorithm to sort an array and print the odd and even elements separately.

I. Introduction

Given an array arr[] of size n, we will use bubble sort to sort this array.

Bubble sort is based on the comparison of two adjacent elements of an array.

II. Algorithm design

The input is stored in the form of an array. We need to sort this array into ascending or descending order.

1. We will take in the total number of entries to sort from the user, then iterate a loop n times to get the entries from the user and store them all in an array.

2. We will define an operation “swap” which interchanges the values of two **adjacent elements** of an array.

3. Suppose we iterate over the array and apply swap operation over two adjacent elements if the element with higher index is smaller than the element with lower index. For an array of size n the **maximum number** of such operations would be n-1.

4. This set of n-1 swaps will bubble up the **highest numbe**r to its right place as the highest element will constantly be swapped until it reaches its corresponding sorted position.

3. Since we are getting **at least** one element at its correct position, we can apply these n-1 swaps, n times over the loop to get the sorted array. This will result in performing at most n*(n-1) swaps.

4. We can do a bit better, as with every loop at least one loop is at its correct position, we don't need to go

through the loop n-1 times, if we have already performed these sets of operations, say i times, we will only need to go through the loop n-1-i times. Now we will iterate the loop,

$$\sum_{i=1}^{n}(n-1-i) = \frac{n^2}{2} - 3\frac{n}{2}$$

times. This is slightly better than our previous attempt.

5. Also, the array may contain already sorted elements, for this we can check if any swaps are taking place or not, if there are no swaps, then we can terminate the loop as there is no point sorting the already sorted array. This will further decrease the runtime of our algorithm.

We will try and dry run the algorithm on a small sample now.

Consider a 5 length array {7,4,5,3,6}

7	4	5	3	6

As 7>4 we will swap them.

4	7	5	3	6

As 7>5, we will swap them.

4	5	7	3	6

As 7>3, we will swap them.

4	5	3	7	6

As 7>6, we will swap them.

We have completed one set of swaps and we can notice that 7 is at it's sorted position, next time we won't check for 7 .i.e the last index.

4	5	3	6	7

As 4<5, no swap will take place.

4	5	3	6	7

As 5>3, we will swap them.

4	3	5	6	7

As 5<6, no swap will take place.

4	3	5	6	7

As 4>3, we will swap them.

3	4	5	6	7

As 4<5, no swap will take place

This is the end of the second set of swaps, and we can see that the array is sorted. We will check if any swaps are taking place, since there won't be any swaps, the loop will terminate.

6. We will print the array by iterating

over it n times. We will print odd and even numbers separately by checking their divisibility by 2. We will print in descending order by looping and taking the indices of the array from n-1 to zero.

III. Time complexity

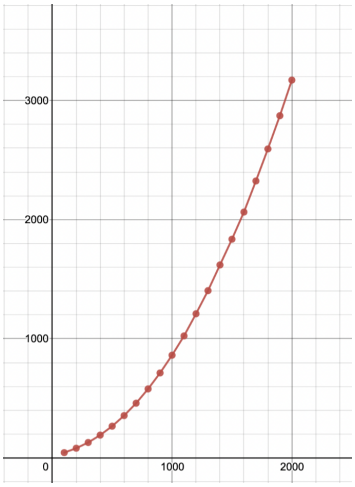
As we know we iterate,

$$\sum_{i=1}^{n} (n-1-i) = \frac{n^2}{2} - 3\frac{n}{2}$$

times, would expect the runtime graph of the loop to look something like x^2 graph.

Input: n, size of array;
100<=n<=2000

Elements in arr[n] have values between 0 and 10000.



Graph of t (in 10^-7 seconds) vs n

This Graph closely resembles the x^2 graph.

N	T|
100	0.000044
200	0.000081
300	0.000129
400	0.000191
500	0.000266
600	0.000355
700	0.000459
800	0.000579
900	0.000713
1000	0.000862
1100	0.001024
1200	0.001210
1300	0.001404
1400	0.001620
1500	0.001836
1600	0.002064
1700	0.002325
1800	0.002594
1900	0.002873
2000	0.003172

The table of n vs t (sec)

IV. Conclusion

Bubble sort is an effective way to sort an array, it has time dependency of n^2.

V. References

- https://www.geeksforgeeks.org
- https://stackoverflow.com

Appendix:

Code:

https://github.com/pvtrpndy/ITP_Assignment_1

Interactive Graph:

https://www.desmos.com/calculator/1dnfe6woos

ITP Project by:

IEC2021063 Javesh Lodha
IEC2021064 Yash Gupta
IEC2021065 Shashi Kumar Chaubey
IEC2021066 Pavitra Pandey

Raw Code:

```c
#include <stdio.h>
int main(){
   printf("Enter the number of elements to sort\n");
   int n;
   scanf("%d", &n);
   int a[n];
   printf("\nEnter the elements to sort\n");
   for(int i=0; i<n; i++){
       scanf("%d", &a[i]);
   }
   int count = 0,c;
   for(int i=1; i<n; i++){
       for(int j=0; j<n-i; j++){
           if(a[j]>a[j+1]){
               c = a[j];
               a[j] = a[j+1];
               a[j+1] = c;
               count=1;
           }
       }
       if(count == 0) break;
       count = 0;
   }
   printf("\nSorted array is: ");
   for(int i=0; i<n; i++){
       printf("%d ",a[i]);
   }
   printf("\nSorted array with even numbers is: ");
   for(int i=0; i<n; i++){
       if(a[i]%2 == 0){
           printf("%d ",a[i]);
       }
   }
   printf("\nSorted array with odd numbers is: ");
   for(int i=0; i<n; i++){
       if(a[i]%2 == 1){
           printf("%d ",a[i]);
       }
   }
   printf("\nSorted array in descending order is: ");
   for(int i=n-1; i>=0; i--){
       printf("%d ",a[i]);
   }
   return 0;
}
```