CS 201 HOMEWORK-2 REPORT
TARIK BUĞRA KARALİ
ID:21703937
SECTION : 2

Logic of Algorihms:
Algorithm 1 creates combined [] size of 2*n and then directly copy the arr1 into combined. Then the int big decide the how many arr2 elements bigger than all arr1 elements.that big number importent because big elements are directly copied combined and do need the flow in nested loop . And then arr2 elements go into nested loops and be decided their appropriate position in combined. After the position is found, combined elements shifted and arr2 element goes into their appropriate index.After the nested loops , big elements are directly copied end of the combined.

Algorithm 2 creates combined [ ] of size 2*n and then pick smaller of current element (current element decided by indexOne and indexTwo) and copy it on combined [ ] next position.
Then check the remaining elements in the arr1 and arr2 , copy the remainings element in combined.

Case i :

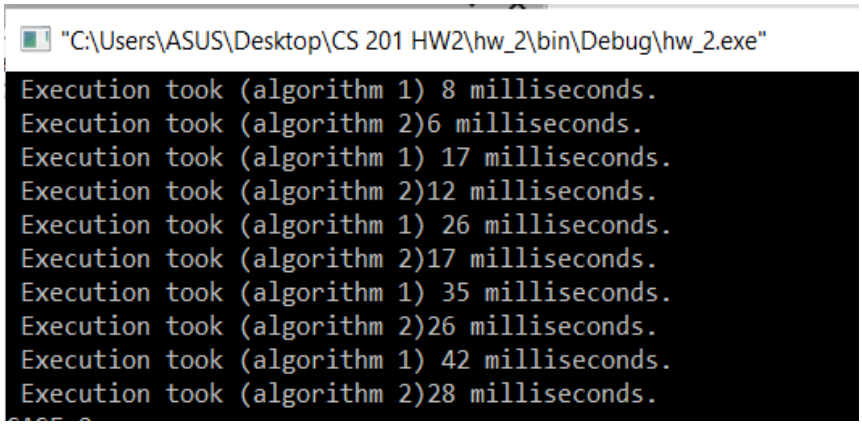
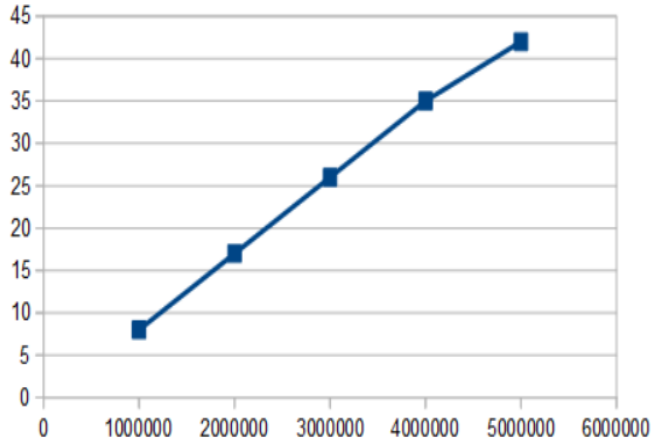


Photo 1

In this case all numbers in *arr1* are smaller than *arr2*. The given values of N is 1000000, 2000000, 3000000, 4000000, 5000000 respectively. Both algorithm is tested with same N values and the results are seen in photo 1 .

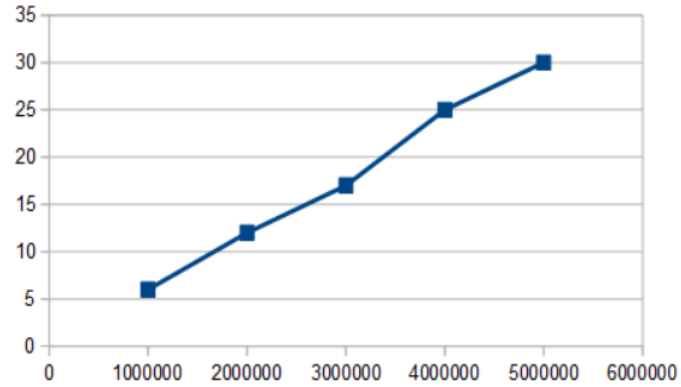
Millisecond



(N)

Graph 1

Milliseconds



(N)

Graph 2

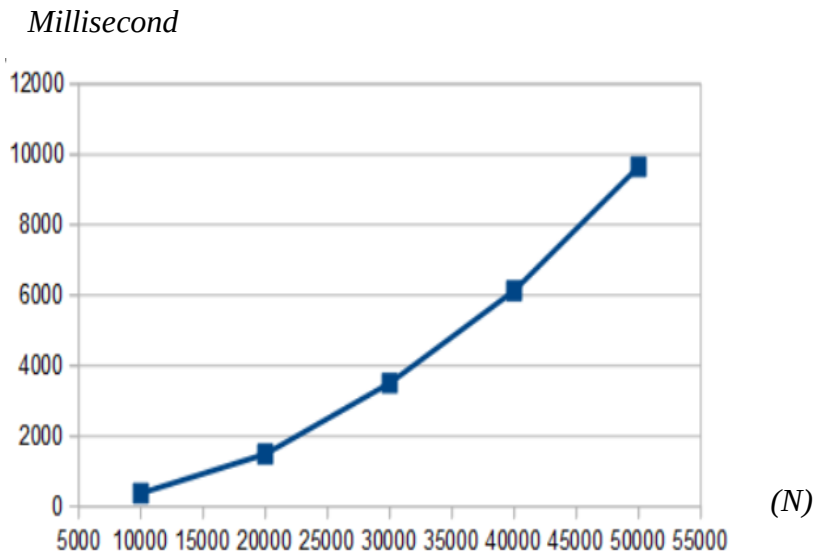
Graph 1 demonstrates a graph that y axis is running time and x axis input size N. Same procedure is accurate for Graph 2. It is observed that both algorith have linear increasing. Theorically algorithm 1 has O(n^2) complexity, but the input arrays in this case are generated all numbers in *arr1* are smaller than *arr2.* So the "big" variable always equals the input size N. Under this condition, algorithm 1 do not need the use part which include nasted for loops . For this reason algorithm 1 behaves O(n) complexity and the graph is linear and this is the best case for algorithm 1. On the other hand algorithm 2 has theorically O(n) complexity. As seen in Graph 2 it has linear line.

Case ii:

```
Execution took (algorithm 1) 381 milliseconds.
Execution took (algorithm 2)5 milliseconds.
Execution took (algorithm 1) 1490 milliseconds.
Execution took (algorithm 2)10 milliseconds.
Execution took (algorithm 1) 3505 milliseconds.
Execution took (algorithm 2)16 milliseconds.
Execution took (algorithm 1) 6130 milliseconds.
Execution took (algorithm 2)22 milliseconds.
Execution took (algorithm 1) 9650 milliseconds.
Execution took (algorithm 2)27 milliseconds.
```

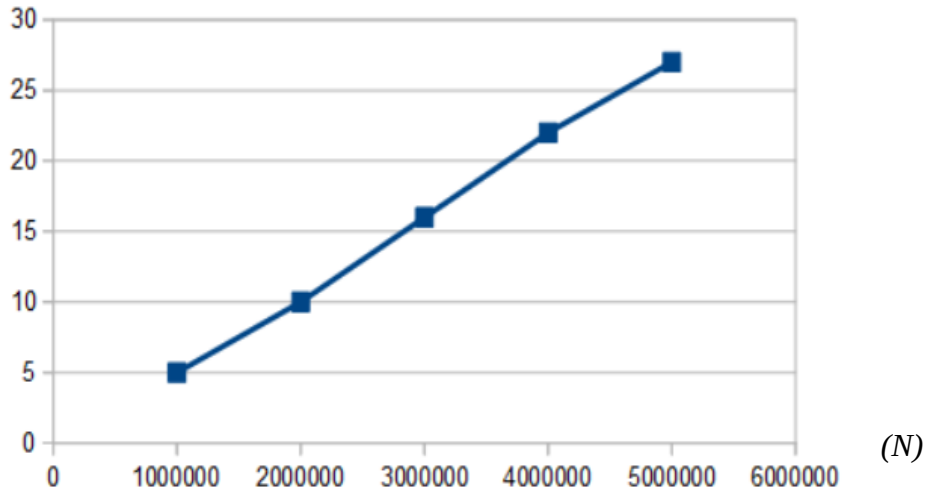
Photo 2

In this case all numbers in *arr2* are smaller than *arr1*.The given values of N for algorithm 1 is 10000, 20000, 30000, 40000, 50000 respectively. The given values of N for algorithm 2 is 1000000, 2000000, 3000000, 4000000, 5000000 respectively. Both algorithm is tested with same N values and the results are seen in photo 2 .



Graph 3

Millisecond



(N)

Graph 4

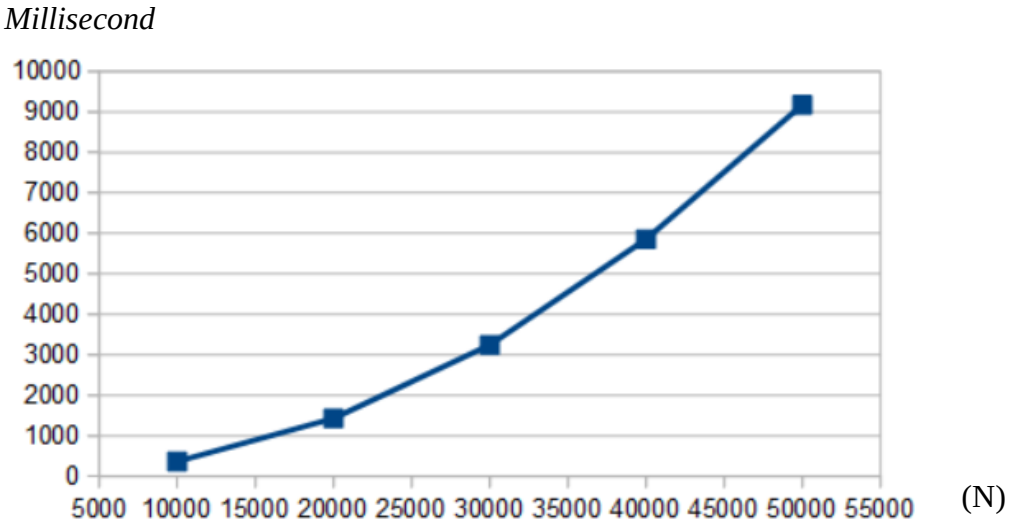
Graph 3 demonstrates a graph that y axis is running time and x axis input size N. Same procedure is accurate for Graph 4. It is observed that both algorith have quadratic increasing. Theorically algorithm 1 has O(n^2) complexity. Since the input arrays in this case are generated all numbers in *arr2* are smaller than *arr1*, the "big" variable not equals the input size N, also it is 0. Under this condition, algorithm 1 need the use part which include nasted for loops to each element. For this reason algorithm 1 behaves O(n^2) complexity and the graph is quadratic and this is the worst case for algorithm 1. On the other hand algorithm 2 has theorically O(n) complexity. As seen in Graph 4 it has linear line.

Case iii:

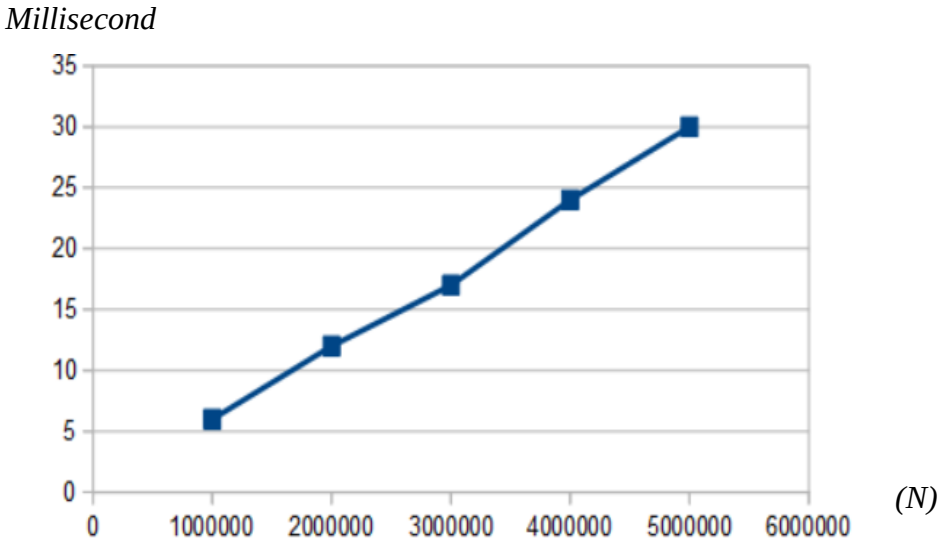
```
Execution took (algorithm 1) 363 milliseconds.
Execution took (algorithm 2)6 milliseconds.
Execution took (algorithm 1) 1428 milliseconds.
Execution took (algorithm 2)12 milliseconds.
Execution took (algorithm 1) 3242 milliseconds.
Execution took (algorithm 2)17 milliseconds.
Execution took (algorithm 1) 5855 milliseconds.
Execution took (algorithm 2)24 milliseconds.
Execution took (algorithm 1) 9173 milliseconds.
Execution took (algorithm 2)30 milliseconds.
```

Photo 3

In this case all numbers in *arr2* and *arr1* are randomly generated.The given values of N for algorithm 1 is 10000, 20000, 30000, 40000, 50000 respectively. The given values of N for algorithm 2 is 1000000, 2000000, 3000000, 4000000, 5000000 respectively. Both algorithm is tested with same N values and the results are seen in photo 3 .

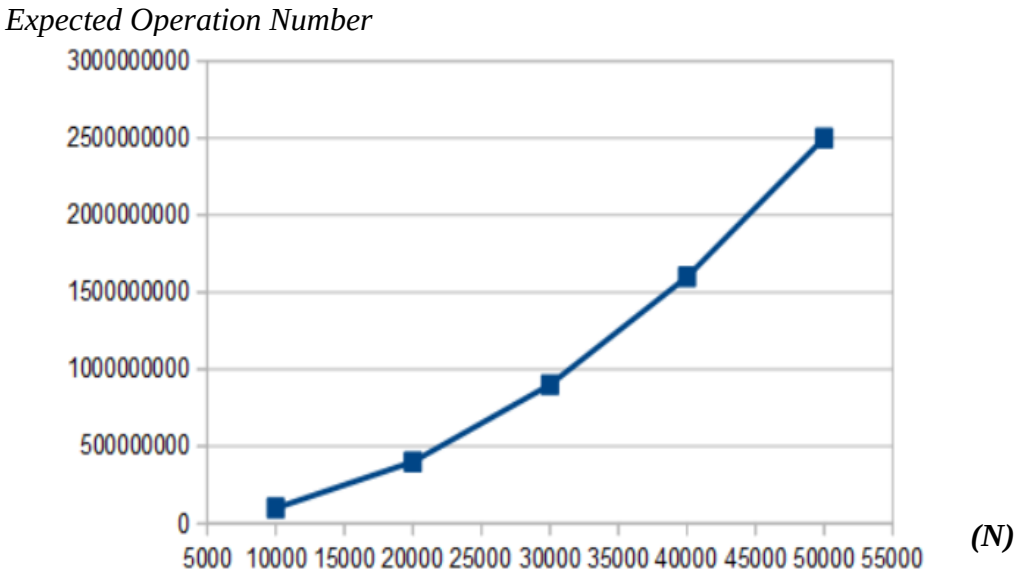


Graph 5

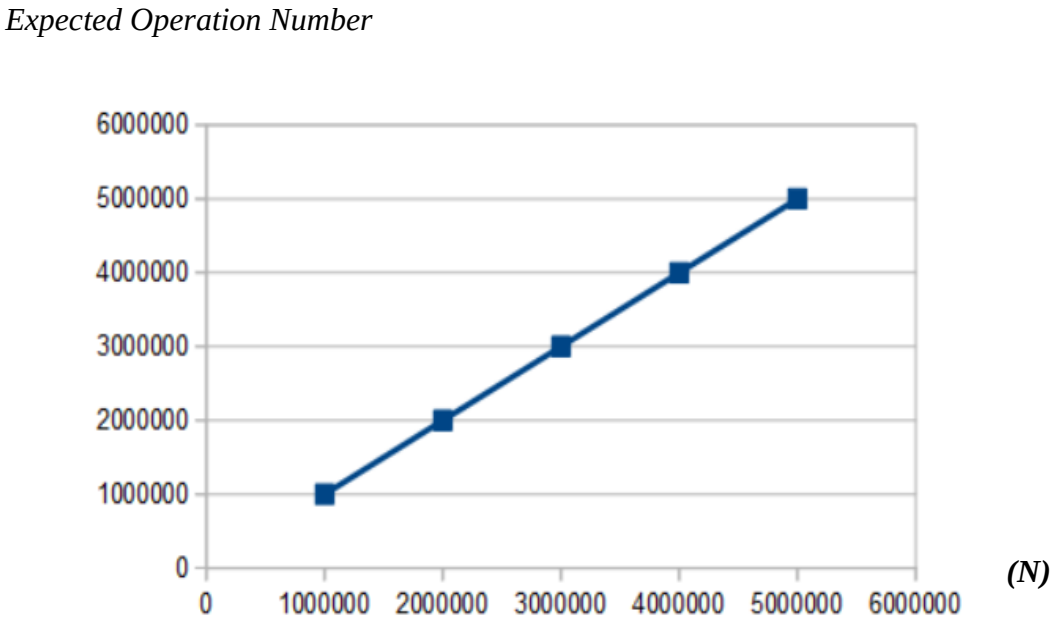


Graph 6

Graph 5 demonstrates a graph that y axis is running time and x axis input size N. Same procedure is accurate for Graph 6. It is observed that algorithm 1 has quadratic increasing. Theorically algorithm 1 has O(n^2) complexity. Since the input arrays in this case are generated numbers in *arr2* and *arr1* randomly, the "big" variable not equals the input size N. Under this condition, algorithm 1 need the use part which include nasted for loops for some elements . For this reason algorithm 1 behaves O(n^2) complexity and the graph is quadratic and this is the average case for algorithm 1. On the other hand algorithm 2 has theorically O(n) complexity. As seen in Graph 6 it has linear line.So the algorithm 2 performs average case.



Graph 7



Graph 8

Graph 7 demonstrates that expected operation number and input size N for the algorithm 1 , Graph 8 demonstrates that expected operation number and input size N for the algorithm 2. According to datas, it is seen that algorithm 2 requires more operation that algorithm 1. With the same operation number, algorithm 2 can read much more input than algorithm 1. So it can be said

that algorith 2 is more efficient than algorithm 2.

Cases:

For algorithm 1 , case i is the best case because of the big number , inputs never flows in nested loop so the complexity is O(n). Case ii is worst case because all the inputs flow in nested loop anda case iii is average case. The time that passed also prove these case's accurancy. On the other hand , for algorithm 2 all the cases gives similar results and they are all average case.

Input N	1000000	2000000	3000000	10000	10000	20000	20000	30000	30000
Case	i	i	i	ii	iii	ii	iii	ii	iii
Time for Algoritm 1 (ms)	8	17	26	381	363	1490	1428	3505	3242

Input N	1000000	1000000	1000000	2000000	2000000	2000000
Case	i	ii	iii	i	ii	iii
Time for Algoritm 2 (ms)	6	5	6	12	10	12

Technical Details:

This algorithms are executed in ASUS FX-550 which has:

- Processor : Intel(R) Core(TM) i7-7700HQ CPU @ 2.80Ghz 2.81Ghz
- RAM : 16.0 GB
- Operating System : 64 bit Operating System , x64 based processor