Homework 2

CSE 246 Analysis of Algorithms, Spring 2015

Due 26.04.2015 Sunday 23:59

Name :İsmet Karakurt

Id :150112030

Name :Burak Balta

Id : 150112039

**Step 1: Designing the Experiment:**

For this experiment,we crafted 30 different arrays for each function to find average case.These 30 different arrays include 10 mostly sorted,10 totaly random,10 mostly reversely sorted arrays and the percent of being sorted changes for each array in order.In brief;

We used for loop and CreateAnArray function to craft 10 different semi-sorted array for insertion sort,such;

For(i=0;i<10;i++){

ArrayType=1;

RLoc=i\*500;

CreateAnArray(Array,SizeOfArray, ArrayType,RLoc);

BaseOp=InsertionSort(Array,Size);

}

In CreateAnArray parameters ,ArrayType selects one of the 3 different array type in the function,which are “totaly random :1” ,”sorted array :2” and ”reversely sorted :3” .RLoc “Random Locations” is used for describing the number of random integer values dropped randomly in to the array on the purpose of changing the percent of being sorted.

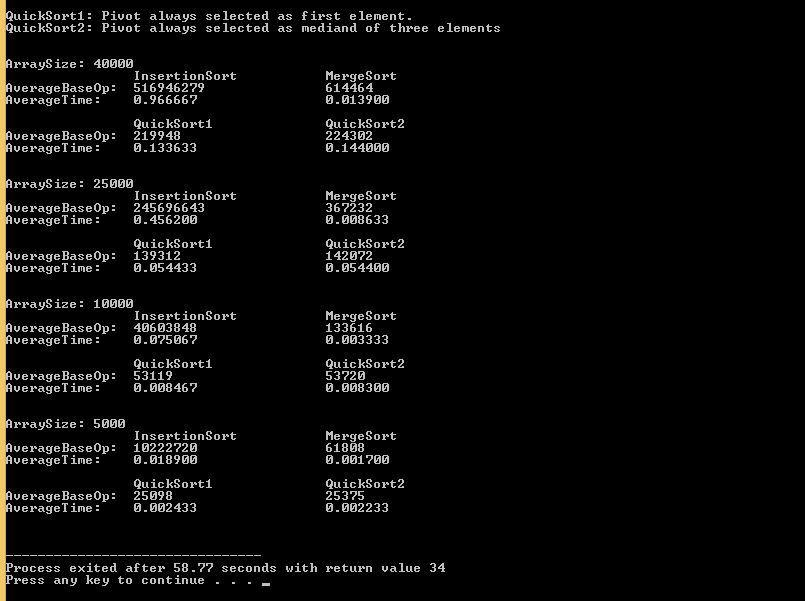
Also,We repeated this process with different array sizes.we used 4 types are about 40k,25k,10k and 5k.

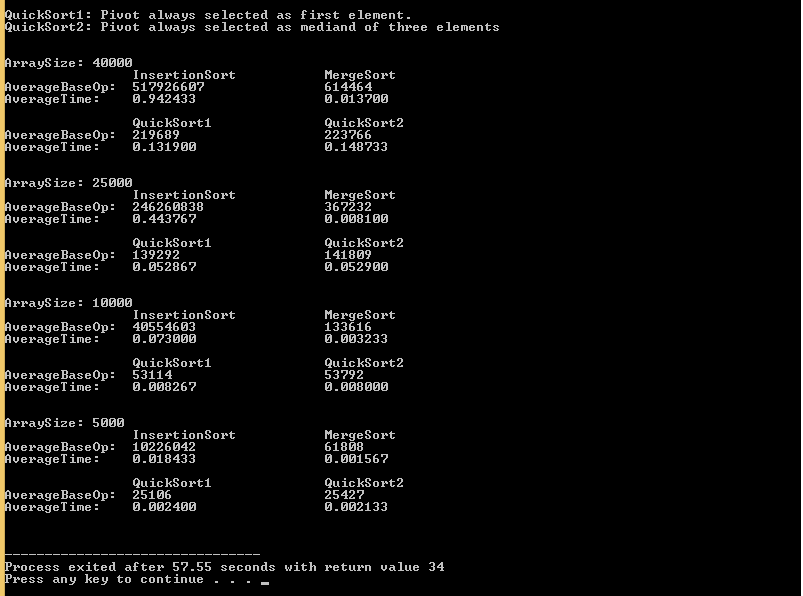
To come up the reasons of these decisions,since the functions we used which have proper worst and best cases ,are quick sort and insertion sort.We wanted to use only the array types that refering to their best and worst cases.The best and worst cases for insertion sort are only about an array to be sorted or not ,or to be reversely sorted.For quick sort,randomly created array is almost same with best case,and the metter of worst case is about again sorted or reversely sorted arrays.

In the code part,some functions have extra code line that influences the time complexity at the very least.So we also wanted to use different array sizes to see the difference between them.For example,median of three pivot selection has more effect on time complexity compared compared to pivot selection as first element.For larger array sizes qucik sort with pivot selected as first element has more adventage than the other.

Besides,We used exact best and worst cases to see how the functions will act.

First,We used totaly random array and totaly sorted array ,totaly reversly sorted array and an array with all elements equal with the size of 5000.Quicksort gives stack overflow when we’r using reversly sorted array ıf we use more than size 5000.

**Step 2: Coding and Running :**These are some outputs of my program; ****

****

****

In the code,we simply runned each function 30 times in a for loop and found the total BaseOperationCount and Time Complexity with adding them in each step of a for loop and then divided them by 30 to calculate average value of each function.

In the output,we showed the average values of time complexity and base operation count for each function and array size.

We counted array movements to calculate base operation count.And for time complexity we called the fucntion clock() just before and after we call our sort fucntion and calculate the difference between them.Like;

CreateAnArray(Array4,Size4,i+3,0);

t1=clock();

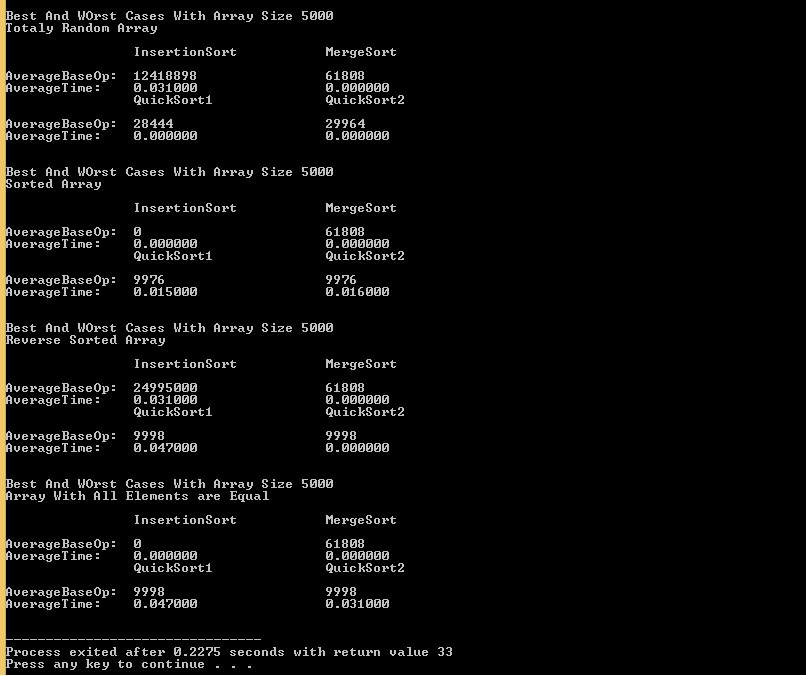
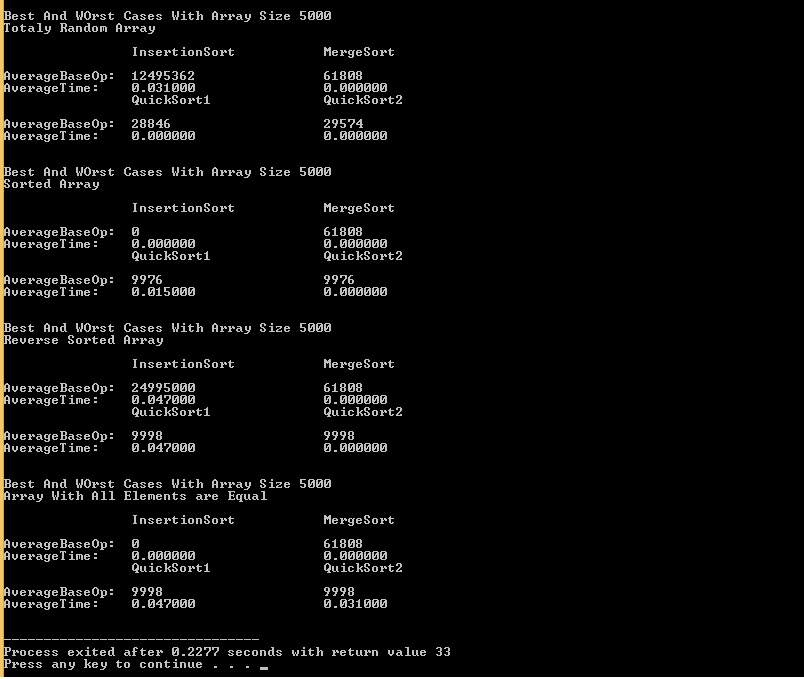
QuickSort(Array4,0,Size4-1,2,&Comparison);

t2=clock();

diff = (((float)t2 - (float)t1) / 1000000.0F ) \* 1000;

TotalTime+=diff;

TotalBaseOp4+=Comparison;

These are some of Best and Worst Case Outputs

**Step 3: Illustrating and Analyzing Results:**

(\*Insertion Sort\*)

Among these functions,insertion sort has the least efficiency even it has the simplest code.We used 10 mostly sorted arrays in our sample but it didnt even affect the result for insertion sort.Algorithm makes an insane numbers of array movement like 9 digit numbers that we calculated for 40k array size.That makes it like ten times slower than the others which makes us to wait like 1 min to see output in each run ☺.In a case like sorted array or array with all elements equal,insertion sort is the best way to sort an array as expected and we saw in the tables.

We can see that BaseOp and TimeComplexity for sorted array and array with all elements equal,is 0 at any table.Because of doing no partition at all,algorithm just does the comparisons and that makes it almost takes no time.

With the array size increasement we can see that baseop does not increase in linear order.

In Output1 table At size 40k BaseOp is like 5Hundred Million

25k BaseOp is like 2 and a half Hundred Million

10k BaseOp is like 40 Million

5k BaseOp is like 10 Million

And changes from Input to another,we can see different values in each outputs…

Compared with the theoretical results that tells us insortion sort is just usefull for sorted arrays,we can confirm our results.

(\*Merge Sort\*)

Merge sort compared to others,has the best case of partition which makes him fastest in almost every case as a theoretical and it is.At every average case and almost all best and worst cases mergesort is the fastest since all cases have same efficiency.Even we make more array movement than any quicksort.

For size 40k in output1;

Time complexity of insertion sort : 0,94

Time complexity of merge sort : 0,013

Time complexity of quick sort1: 0,131

Time complexity of quick sort2: 0,148

Again since all cases have same efficienc BaseOperationCount didnt change for the same array sizes.if we come to time complexity,results are allmost the same like;

0,0135 0,0137 0,0139

Because of using randomly generated arrays,we realised,in such a case of array to be more smilar to sorted array ,merge sort spends less time to transfer residual part of array and that makes function a bit faster.So we can say it can be a best case for mergesort at least.

But the fact of using temporary arrays limits our array size unlike others.We couldnt use more then 50 k sized array with mergesort while we were able to use 300000 with insertion sort and like 80000 with quick sort.

(\*Quick Sorts\*)

About using randomly generated arrays,quick sorts are much more better than insertion sort and has almost smilar results with merge sort.If we look into best and worst case outputs randomly generated arrays for 5k size quick sorts have 0 time complexity like mergesort.

Quick sort with the pivot which selected as first element and quick sort with the pivot which selected as median of three are almost same.But as it explained theoreticaly,sorted array is worst case for quick sort 1 and also this is the reason that we use median of three pivot selection.If we use sorted or reversly sorted array as input quick sort 1 is acts as slow as insertion sort cause of failure of the partition case.In other case quick sort 2 isnt affected that much from this situation.

In average cases quick sorts are almost smilar with eachother and while better than insertion sort ,they are not as effective as merge sort.

For size 40k in output1;

Time complexity of insertion sort : 0,94 517 Milion Base Ops

Time complexity of merge sort : 0,013 614464 Base Ops

Time complexity of quick sort1: 0,131 219689 Base Ops

Time complexity of quick sort2: 0,148 223766 Base Ops

Quick sort algorithms have absolute advantage for base operation case and memory allocation compared to merge sort.In other case,while calculating average case we used 10 mostly sorted and reversly sorted arrays which greately decreases the performance of quick sort.

On the other hand quick sort 2 has extra line of code for selecting pivot that increases time complexity.In higher size arrays quick sort array 2 lasts longer than quick sort 1.

In some cases of more arrays to be close to best case of quick sort (split in the middle),quick sort 1 can be more effective than quick sort 2 even for lower array sizes.

In Output1

For size 40k

Time complexity of quick sort1: 0,131

Time complexity of quick sort2: 0,148

For size 25k

Time complexity of quick sort1: 0,0528

Time complexity of quick sort2: 0,0529

For size 10k

Time complexity of quick sort1: 0,0082

Time complexity of quick sort2: 0,0080

For size 5k

Time complexity of quick sort1: 0,0024

Time complexity of quick sort2: 0,0021