Enes Bedirhan Dikmen

150180016

Homework 1 Report

Part1

I have written my implementations in "myquicksort.h" file besides i/o operations.

My main.cpp file can be compiled with:

g++ -std=c++11 -Wall -Werror main.cpp

And can be run with:

./main.out N

Part2

a) In partition process as we go from left to right for each element of N we check if they are smaller than the pivot and swap the positions of the ones needed. Hence time taken for partition (cn):

Θ(N)

Best case: The best case for quicksort is when pivot is choosen to be middle element of the array which means the last index of the array for my implementation. If this is the case for each partition the recurrence equation would be:

T(N) = 2T(N/2) + Θ(N)

To use master theorem for the form "T(n) = aT(n/b) + f(n)" a = 2, b=2, and f(n) = Θ(n). Than $log_b\, a$ = 1, which also equal c in Θ(n^c) . We will use case 2: Θ($n^c\, log\, n$):

Θ($nlog\, n$)

Worst case: If we would pick the pivot to be biggest or the smallest element in each recurrence there would only one sub-array to be partitioned. One sub-array is size 0 and other the actual one's size is N-1. Recurrance equation for this one:

T(N) = T(0) + T(N-1) + Θ(N) (T(0) = Θ(0) = 0)
T(N) = T(N-1) + Θ(N)

If time taken for partition for N elements is cN, as we go down this number goes like c(N-1), c(N-2), c(N-3)… If we sum this up we get c((n+1)(n/2)-1) = c(N^2/2 + N/2 – 1) = (cN^2/2 + cN/2 – c). So:

Θ(n^2)

Average case: Since heavy duty of algorithm is done in partitioning and comparisons in it, calculating total number of comparisons is good way to get expected running time. This can be expressed as:

$\sum_{a=1}^{n} \sum_{b=(a+1)}^{n} X_{a,b}$ Where Xa,b = 1 if elements a and b are ever compared, 0 if not.

$\sum_{a=1}^{n} \sum_{b=(a+1)}^{n} \frac{2}{b-a+1}$ 2/(b-a+1) = probability of 2 elements comparison.

This equals lower than 2n ln(n). This means average running time :

O(n log(n))

b)

1) Sort the sales.txt data by the total profits and write it into sorted_by_profits.txt

2) Sort the sorted_by_profits.txt data according to country names using QuickSort Does this solution give us the desired output for all cases?

No, this will not give us desired outputs.

1. Because in quicksort in each recurrance we sort target elements according to choosen pivot. This means depending on the pivot selection we could swap an element's position and if another element with same value exist we could change their relative positions. An example for this in our dataset:
   Here we already have sorted_by_profits.txt and now we are applying quicksort by their country names. Pivot choosen to be rightmost element as it is in my implementation.

   Our blue and red bars initialized(The elements below the bars are their pointing location):

Tanzania	Cosmetics	739008080	7768	1350622.16
Slovakia	Beverages	174590194	3973	62217.18
Tanzania	Beverages	659878194	1476	23114.16
South Africa	Fruits	443368995	1593	3839.13

Blue bar increments until something smaller than pivot (Slovakia < South Africa):

Tanzania	Cosmetics	739008080	7768	1350622.16
Slovakia	Beverages	174590194	3973	62217.18
Tanzania	Beverages	659878194	1476	23114.16
South Africa	Fruits	443368995	1593	3839.13

Blue bar element and red bar element are swapped ther they both incremented by 1:

Slovakia	Beverages	174590194	3973	62217.18
Tanzania	Cosmetics	739008080	7768	1350622.16
Tanzania	Beverages	659878194	1476	23114.16
South Africa	Fruits	443368995	1593	3839.13

Blue bar incremented until something smaller than pivot. Nothing is smaller ahead so goes until the pivot:

Slovakia	Beverages	174590194	3973	62217.18
Tanzania	Cosmetics	739008080	7768	1350622.16
Tanzania	Beverages	659878194	1476	23114.16
South Africa	Fruits	443368995	1593	3839.13

Pivot swapped with red bars position to put the pivot to correct place:

Slovakia	Beverages	174590194	3973	62217.18
South Africa	Fruits	443368995	1593	3839.13
Tanzania	Beverages	659878194	1476	23114.16
Tanzania	Cosmetics	739008080	7768	1350622.16

Pivot swapped with red bars position to put the pivot to correct place and partition is over.
After this the left recurring quicksort won't make any change since it has only one element. And the right recurring quicksort won't make any change too because two elements (Tanzania = Tanzania) have the same value so there is no swaps. This means we have this resulting list:

Slovakia Beverages 174590194 3973 62217.18

South Africa Fruits 443368995 1593 3839.13

Tanzania Beverages 659878194 1476 23114.16

Tanzania Cosmetics 739008080 7768 1350622.16

As we can see above last two element's order by total profit is not preserved due to unstable nature of quicksort.

2. We need stable sorting algorithms that will preserve the relative order of elements with same values. These could be:

Merge Sort

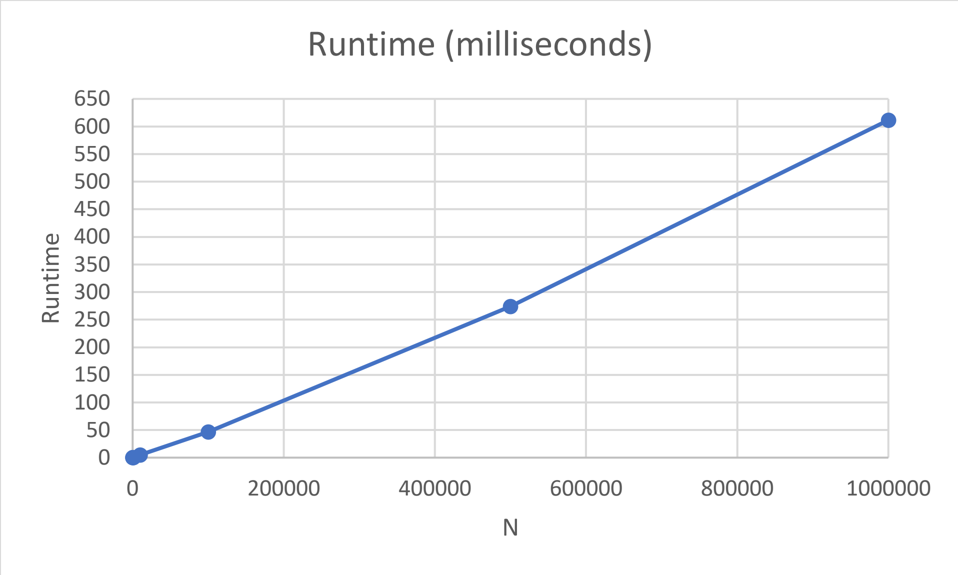
Insertion Sort

Counting Sort

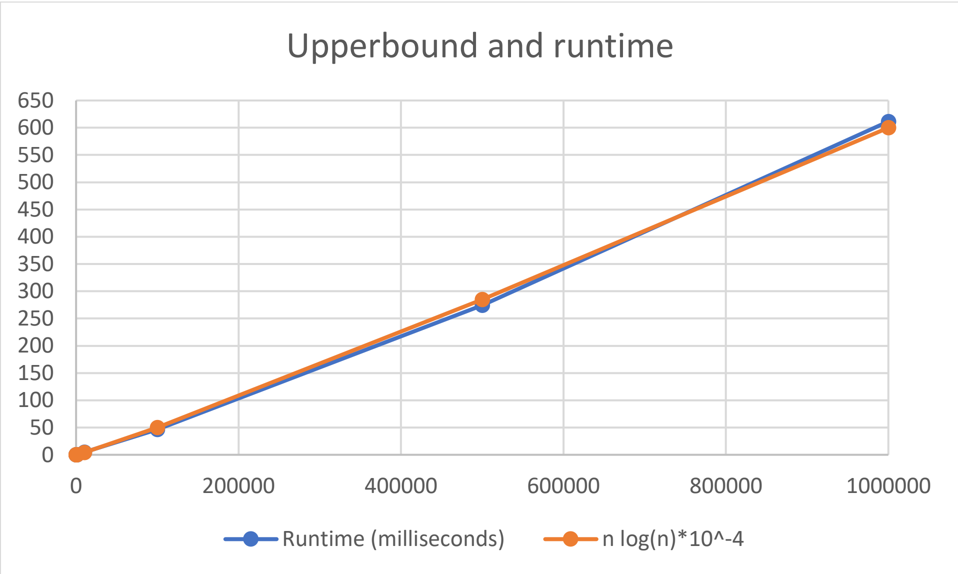
c. I have done all of the runtime measurements in ITU SSH server.

Table of averige runtimes for different input sizes in milliseconds:

N =	10	100	1000	10K	100K	500K	1M
Average runtime	10.59 x 10^-4	13.56 x 10^-3	0.195	2.832	40.72	274.2	611.5



Nlog(n) values goes like 10, 200, 3000… and my runtimes goes like 0.00159, 0.01356, 4,832 … So applying a factor of 10^-4 will get the values closer to compare. 10^-4 * n log(n) = O(nlog(n))



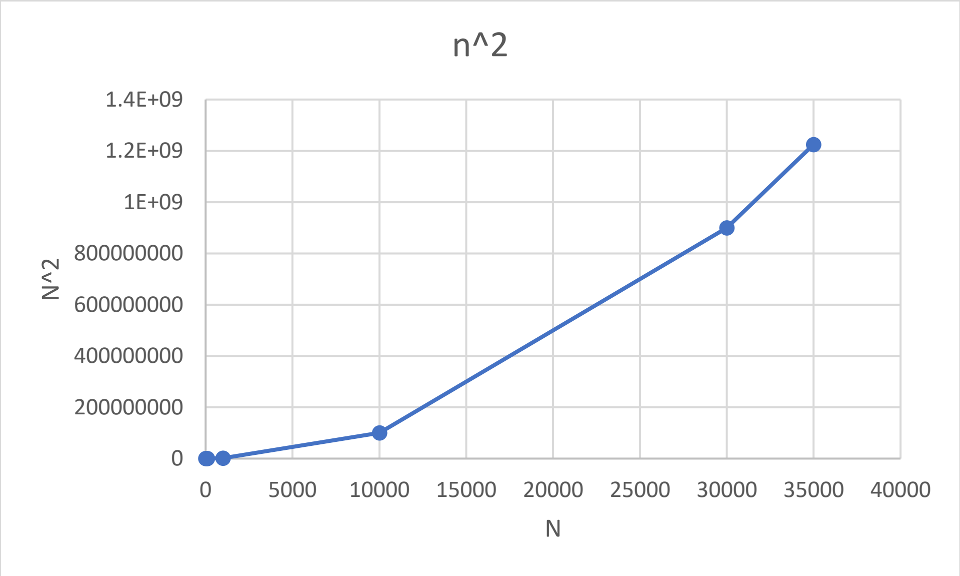
As the graph shows average runtimes and n log(n)*10^-4 values are very close to each other. This means my program follows the asymptotic upper bound with a constant of approximately 10^-4.

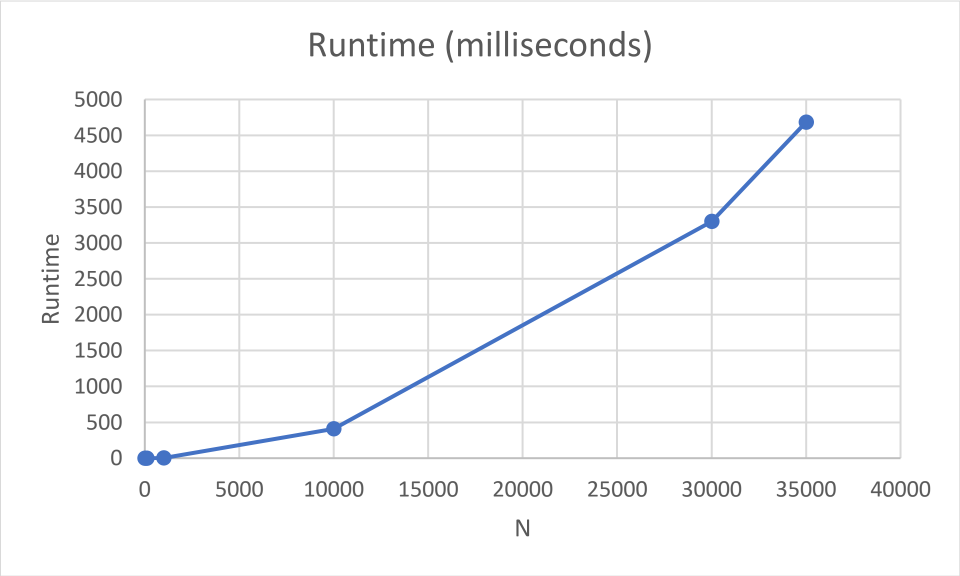
d)

1.Since this case took to long to measure at ssh i measured this on my computer. Even this way the highest N value i was i able to measure the average runtime was 35K.

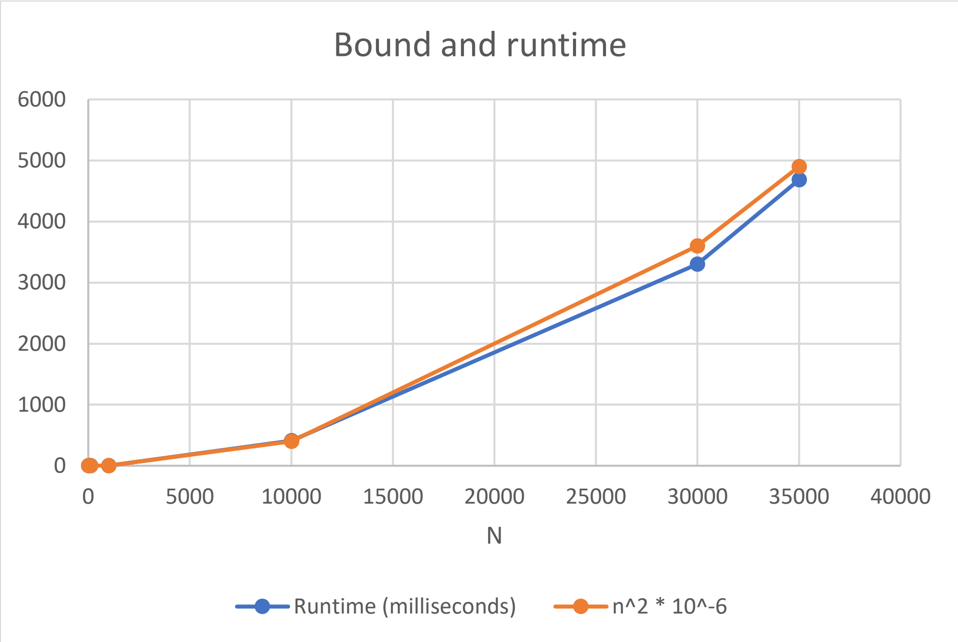
Average runtimes on “sorted.txt” in milliseconds.

N =	10	100	1000	10K	30K	35K
Average runtime	21.88 x 10^-4	98.05 x 10^-3	3.531	407.6	3307	4684





As shown in the graphs runtime and Θ(n^2) have similar patterns. If we merge them into one graph and scale n^2 with 4*(10^-6) we obtain this graph(4*(10^-6)*n^2 = Θ(n^2)):



The reason for this huge increase in runtime is working on “sorted.txt” we have the worst case scenario. Since pivot is choosen rightmost element it is always the biggest element. This causes the recursion number to rise to N^2. This is why i wasn’t able to measure the average time taken on this case. A 10 times increase in input size causes 100 times incerase in runtime, which would have been a number slightly bigger than 10 for the average case for most cases.

2.An input of sales ordered in exact reverse order of “sorted.txt” would give the similar results.

3.Chosing the pivot randomly or at middle point will solve this.