QuickSort

Tested in

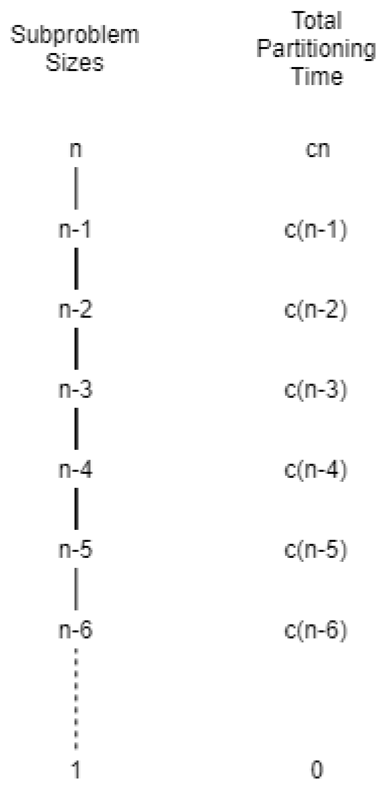
```
g++ version: 4.9.2 (windows 10)
g++ version: 4.8.5 - ITU SSH
```

Running Commands

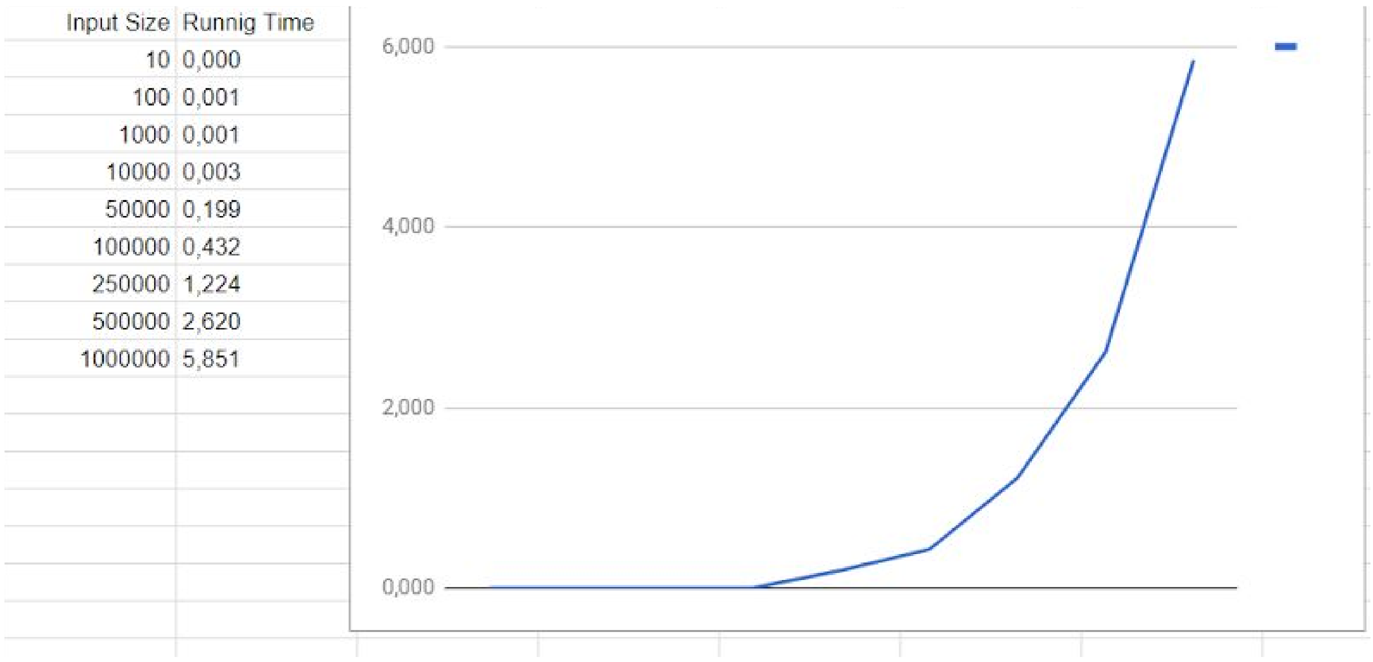
```
g++ main.cpp -o main.exe
./main.exe input_size
```

a) T(N) = the worst case running time for N elements.

The worst case in quicksort occurs when the dataset is already sorted. Because we select our pivot as left-most or right-most element and if dataset is already sorted, our pivot value will be smallest or largest element in the dataset. As a result of this situation, imbalanced partitioning will happen in every recursion and one of the our subarrays will be empty every time.

T(N) = T(0) + T(N-1) + θ(N) =θ(1) + T(N-1) + θ(N) = T(N-1) + θ(N) =θ(N^2) If we sum up all of the arithmetic series cn+c(n−1)+c(n−2)+⋯+2c =c(n+(n−1)+(n−2)+⋯+2) =c((n+1)(n/2)−1) Then we will ignore low order terms and constants when using big-θ notation. Worst runnig time is θ(N^2)	 Subproblem Sizes Total Partitioning Time n cn n-1 c(n-1) n-2 c(n-2) n-3 c(n-3) n-4 c(n-4) n-5 c(n-5) n-6 c(n-6) 1 0

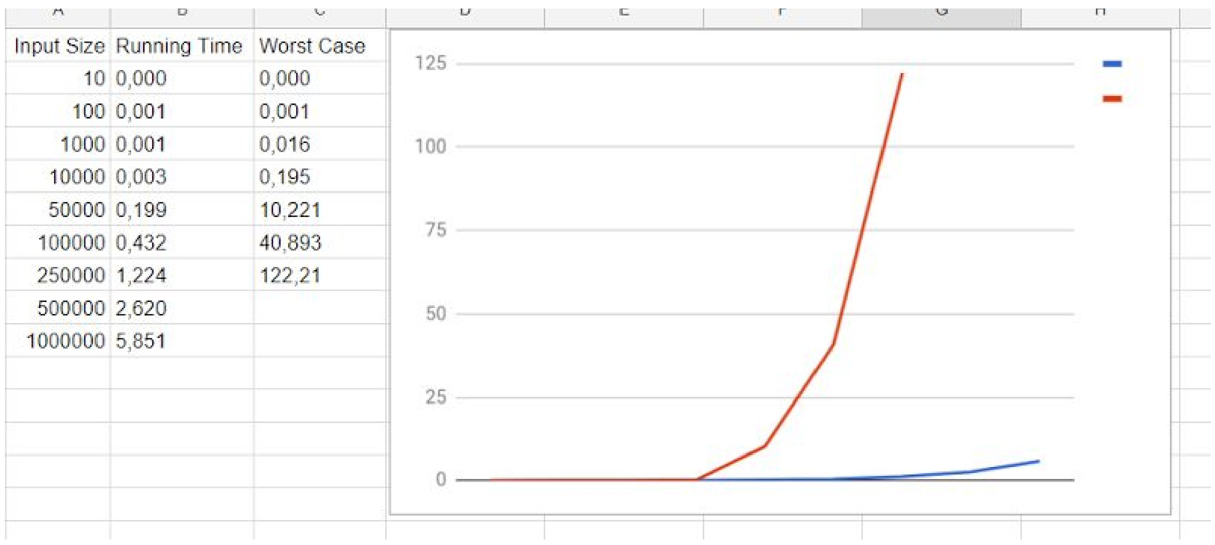
b) A balanced partitioning will be best case for us. Which means left and right subarrays have N/2 where N is the number of elements in original array. Both these quicksort calls in best case partitioning will take time T(N/2). So T(N) for best case will be T(n) = 2T(N/2) + aN
If we apply the master method a=2, b=2. case2: T(N)=θ(nlogn)



Input Size	Runnig Time
10	0,000
100	0,001
1000	0,001
10000	0,003
50000	0,199
100000	0,432
250000	1,224
500000	2,620
1000000	5,851

Backup link if you cannot view properly

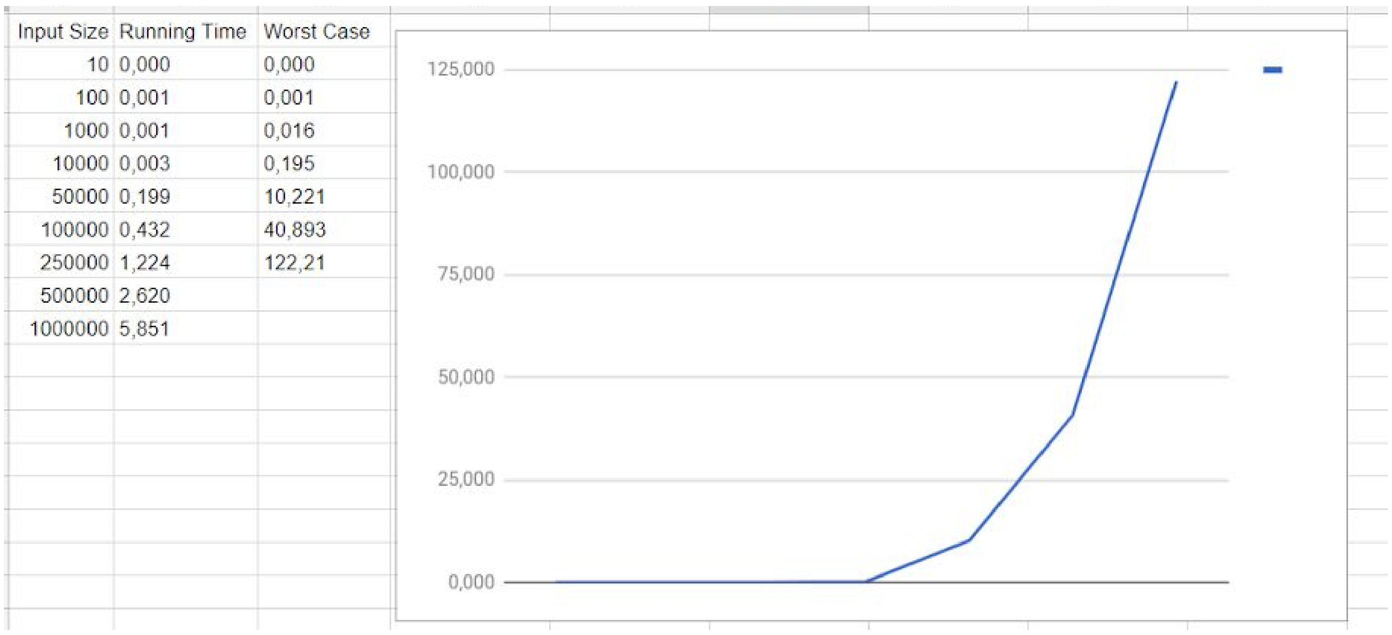
Comparative Graphs for worst case - average case.



Input Size	Running Time	Worst Case
10	0,000	0,000
100	0,001	0,001
1000	0,001	0,016
10000	0,003	0,195
50000	0,199	10,221
100000	0,432	40,893
250000	1,224	122,21
500000	2,620	
1000000	5,851	

c) The worst case in quicksort occurs when the dataset is already sorted in ascending or descending order. Because we select our pivot as left-most or right-most element and if data set is already sorted, our pivot value will be smallest or largest element in the dataset. As a result of this situation, imbalanced partitioning will happen in every recursion and one of the our subarrays will be empty every time.

To construct worst case, I have exported all dataset in ascending order and applied quicksort algorithm to exported .csv file. (I commented out exporting function calls in source code.)



Input Size	Running Time	Worst Case
10	0,000	0,000
100	0,001	0,001
1000	0,001	0,016
10000	0,003	0,195
50000	0,199	10,221
100000	0,432	40,893
250000	1,224	122,21
500000	2,620	
1000000	5,851	

Time complexity for quicksort is O(nlogn) in best or average case and O(n^2) in worst case. But worst case can be avoided by using randomized version of quicksort.

Other possible solutions to avoid the worst cases

1. Choosing the middle index of the partition as a pivot.
2. Choosing the median of the first, middle and last element of the partition for the pivot.

d) Quicksort is not a stable sorting algorithm. In a stable sorting algorithm, the relative order of records with equal key is preserved.

For example: assume that we want to sort a list of points in cartesian plane. Each records in the list here is a pair of integers, first integer is X-coordinate and second integer is Y-coordinate. If we want for sort in increasing order of X-coordinates. We have two records with equal X-coordinate. If we will use a stable sorting algorithm then (4,5) that is before point (4,3) in the original list will be before (4,3) in sorted arrangement also. But this is not guaranteed if we will use an unstable algorithm like quicksort. In the second arrangement elements are sorted in increasing order of X-coordinate but (4,3) is coming before (4,5).

```
Points in cartesian plane:
(1,2) (4,5) (2,3) (4,3) (5,2)
(1,2) (2,3) (4,5) (4,3) (5,2)
or
(1,2) (2,3) (4,3) (4,5) (5,2)
```

Partitioning logic does not ensure stability in quicksort algorithm.

For my quicksort implementation: These two datas printed out in different orders in executions.

```
10 , 70 , 74 , female , 74565 , 8600000US74565
10 , 25 , 29 , female , 97843 , 8600000US97843
10 , 5 , 9 , male , 99778 , 8600000US99778
11 , 0 , 4 , female , 4675 , 8600000US04675
11 , 18 , 19 , male , 5669 , 8600000US05669
11 , 75 , 79 , male , 12412 , 8600000US12412
11 , 20 , 20 , female , 12865 , 8600000US12865
11 , 10 , 14 , male , 13162 , 8600000US13162
11 , 35 , 39 , male , 17934 , 8600000US17934
11 , 62 , 64 , female , 17934 , 8600000US17934
11 , 25 , 29 , male , 24139 , 8600000US24139
11 , 80 , 84 , male , 29584 , 8600000US29584
11 , 18 , 19 , male , 42236 , 8600000US42236
11 , 10 , 14 , female , 45032 , 8600000US45032
11 , 40 , 44 , female , 52225 , 8600000US52225
11 , 15 , 17 , male , 59241 , 8600000US59241
11 , 75 , 79 , female , 59925 , 8600000US59925
11 , 65 , 66 , male , 60928 , 8600000US60928
11 , 85 , 0 , female , 62421 , 8600000US62421
11 , 65 , 66 , male , 72351 , 8600000US72351
11 , 25 , 29 , male , 76827 , 8600000US76827
12 , 55 , 59 , female , 12416 , 8600000US12416
12 , 18 , 19 , male , 14720 , 8600000US14720
```