Algorithm Analysis
CSI 5301

Assignment 3.1: How to Pick a Pivot
Due: Mon May 30, 2022 11:59pm
Submitted by: Pawan Bhandari

Description:
Apply the following rules for picking a pivot to sort Integers using the quick sort algorithm and analyze the number of comparisons required for each strategy.

Pivoting Rules

1. Using the **first** element of the array as the pivot element.
2. Using the **final** element of the given array as the pivot element.
3. Using a **random** element of the given array as the pivot element.
4. Using the "**median-of-three**" pivot rule.

Data Structure:
vector<int>

Algorithm:
1) Store the numbers in the input file as the elements of vector

populateInputVector(vector<int>* vector, string file)

2) Function to determine the median value and return its index

pickMedianOfThree(vector<int>* vector, int l, int r)

start
middleIndex = (r – l + 1) / 2 ,when number of integers is odd
middleIndex = ((r – l + 1) / 2) - 1 , when number of integers is odd
if (final <= first <= middle) or (final >= first >= middle) return index of first
if (first <= middle <= final) or (first >= middle >= final) return index of middle
if (first <= final <= middle) or (first >= final >= middle) return index of final
end

3) Function to pick a pivot based on the pivot rule

findPivot(vector<int>, int l, int r, string pivotRule)

start
if (pivotRule == "first") return l
if (pivotRule == "final") return r
if (pivotRule == "random") return random number between l and r
if (pivotRule == "median") call **pickMedianOfThree** and return median index
end

4) Function to partition the unsorted list

partition(vector<int>* vector, int left, int right, string pivotRule)

start
call **findPivot** function to set *pivotIndex;*
pivot = vector.at(pivotIndex);
swap elements at *pivotIndex* and left;
int *i* = left +1
for *j* = left + 1 to right
if (element at j < pivot) {
swap elements at *j* and *i*;
i++;}
swap elements at *l* and *i – 1;*
return *i - 1* as sorted position of the pivot element;
end

5) QuickSort algorithm

quickSort(vector<int>* *vector*, int *left*, int *right*, string *pivotRule*, int& *count*)

start
if l*eft >= right* return;
count = count + (right – left);
sortedPivotIndex = ***Partition(****vector, left, right, pivotRule****);***
quickSort*(vector, left, sortedPivotIndex – 1, pivotRule, count);*
quickSort*(vector, sortedPivotIndex + 1, right, pivotRule, count);*
end

Analysis:

Input N	vector<int>
Basic Operation	Basic arithmetic operations Vector operations Recursion
Recurrence relation	Best Case: $T(n)=2T\left(\frac{n}{2}\right)+O(n)$ Worst Case: $T(n)=T(n-1)+O(n)$

Worst Case Analysis:

Worst case occurs when a pivot is picked such that it is always the smallest element in the list. When this happens the list is split into two halves 1 and n-1 size in each recursive call. Since partition takes O(n) time, we can write a recurrence relation for worst case and solve by substitution to derive the worst case complexity as follows:

$$T(n)=T(n-1)+n$$

or, $T(n)=\left[T(n-2)+(n-1)\right]+n$

or, $T(n)=T(n-2)+(n-1)+n$

or, $T(n)=\left[T(n-3)+(n-2)\right]+(n-1)+n$

if we repeat this back-substitution for k^{th} time, we get

$$T(n)=T(n-k)+(n-(k-1))+(n-(k-2))+(n-(k-3))+.....+(n-1)+n$$

assume that $n-k=0$, then n = k. so,

$$T(n)=T(n-n)+(n-n+1)+(n-n+2)+(n-n+3)+.....+(n-1)+n$$

or, $T(n)=T(0)+1+2+3+.....+(n-1)+n$

or, $T(n)=T(0)+\sum_{1}^{n}1+2+3+.....+n$

or, $T(n)=T(0)+n\frac{(n+1)}{2}$

so, $T(n)=O(n^2)$

Therefore, worst case for QuickSort algorithm is $O(n^2)$

Best Case Analysis:

Best case occurs when pivot is picked such that it is always the median of the elements in the list. When this happens the list is split into two equal halves each time in each recursive call. Since partition takes O(n) time, we can write a recurrence relation for worst case and apply Master Theorem to derive time complexity as follows:

$$T(n)=T\left(\frac{n}{2}\right)+T\left(\frac{n}{2}\right)+n$$

or, $T(n)=2T\left(\frac{n}{2}\right)+n$ Comparing this recurrence relation with $T(n)=aT\left(\frac{n}{b}\right)+O(n^d)$ we get,

$a = 2, b = 2, d = 1$. so, $log_2 2 = 1 = d$ and in this case $O(n^d \log n)$

Therefore, best case for QuickSort algorithm is $O(n \log n)$

Comparison Results:

QuickSort algorithm was applied to the given list of integers between 1-1000 using the four pivot strategies and the comparisons done for each run is listed in the table below.

Table 1: Comparison count for different pivot strategies

	First	Final	Random	Median-of-three
QuickSort10k	162085	164123	149111	138382
QuickSort10kR	49995000	49995000	164057	12505002
QuickSort10kSemi2k	229376	209217	156915	136927
QuickSort10kSemi4k	200624	220313	159336	132119

Table 2: Comparison count for first pivot strategy on different runs

	First-1	First-2	First-3	First-4	First-5
QuickSort10k	162085	162085	162085	162085	162085
QuickSort10kR	49995000	49995000	49995000	49995000	49995000
QuickSort10kSemi2k	229376	229376	229376	229376	229376
QuickSort10kSemi4k	200624	200624	200624	200624	200624

Table 3: Comparison count for final pivot strategy on different runs

	Final-1	Final-2	Final-3	Final-4	Final-5
QuickSort10k	164213	164213	164213	164213	164213
QuickSort10kR	49995000	49995000	49995000	49995000	49995000
QuickSort10kSemi2k	209217	209217	209217	209217	209217
QuickSort10kSemi4k	220313	220313	220313	220313	220313

Table 4: Comparison count for random pivot strategy on different runs

	Random-1	Random-2	Random-3	Random-4	Random-5
QuickSort10k	149111	164035	151468	164961	151540
QuickSort10kR	164057	148404	151468	154140	153550
QuickSort10kSemi2k	156915	149474	143334	148835	157768
QuickSort10kSemi4k	159336	147547	160029	150359	151372

Table 5: Comparison count for median pivot strategy on different runs

	Median-1	Median-2	Median-3	Median-4	Median-5
QuickSort10k	138382	138382	138382	138382	138382
QuickSort10kR	12505002	12505002	12505002	12505002	12505002
QuickSort10kSemi2k	136927	136927	136927	136927	136927
QuickSort10kSemi4k	132119	132119	132119	132119	132119

Following are the observations based the comparison data from tables above:

1) Comparisons are highest for sorting the (reversely) sorted list for both first and final element as Pivot

   This seems to run on $O(n^2)$ *for n=10k, comparisons are in the range of* n^2

2) Comparisons when using random element as Pivot is less in all cases.

   This seems to run on $O(n \log n)$ *for n=10k, comparisons are in the range of* $n \log n$.

   $1000 \times \log 10000 \approx 132877$

3) In cases where the list is not initially sorted, partitioning the list using any of the pivot strategies yields similar comparisons which is almost in the order of *nlogn*

Figures 1 & 2 below depict the number of comparisons for the input files while using different pivot strategies.
It can be observed from the Figure 1 below that the comparisons while using the given four pivot strategies is negligible (hence, not visible in the stacked column chart) when the list is not sorted to start with as compared to the comparisons while sorting already sorted list (worst case).

Figure 2 depicts the comparisons when picking a random element as pivot and it can be seen that the number of comparison is an order of *nlogn*. As we can see from the horizontal line for *nlogn* in figure 2 that comparison count is just a bit higher than the best case of nlogn.

With this data and analysis its evident that QuickSort using random element as pivot runs with time complexity similar to *O(nlogn)*.

Figure 1: Comparisons during QuickSort using different pivot strategies

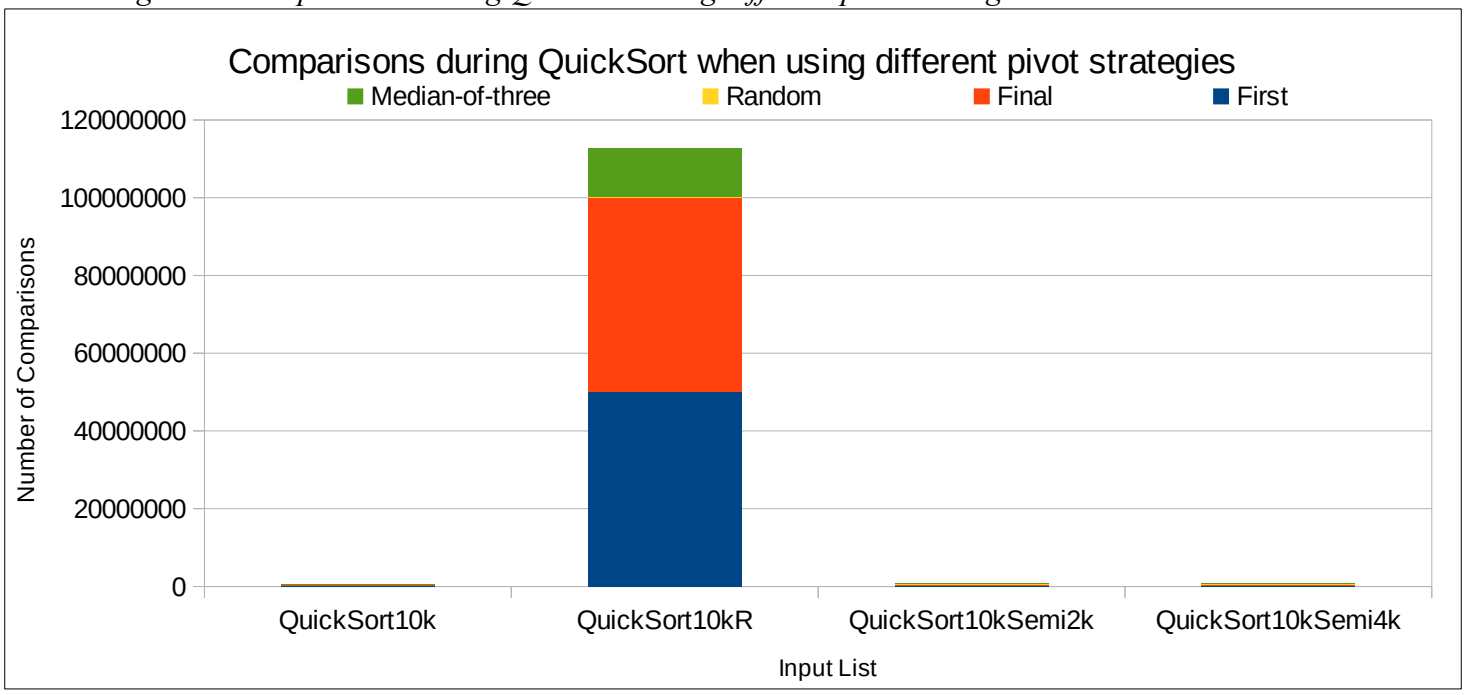

Figure 2: Comparisons during QuickSort using random pivot strategy