Chuan Lin PA3

Menu

Natural Language Description of Quicksort using Median-of-three Partition

a.Description of Quicksort

As decribed in CLRS textbook, page 183, quicksort process, like merge sort, applies the divide-and-conquer method. Here is the three-step divide-and-conquer process for sorting a sub-array A[p:r].

Divide: by partitioning (rearranging) the array A[p:r] into two (possibly empty) subarrays A[p:q-1] (the low side) and A[q:r] (the high side) such that each element in the low side of the partition is less than or equal to the pivot A[q], which is, in turn, less than or equal to each element in the high side. Compute the index q of the pivot as part of this partitioning procedure

Conquer: by calling quicksort recursively to sort each of the subarrays A[p:q-1] and A[q+1:r]

Combine: by doing nothing.

b.Description of Partition

As described in CLRS textbook, page 183, partition process rearranges the subarray A[p,r] in place, returning the index of the dividing point between the two sides of the partition.

c.Description of Median-of-three Partition

As described in PA3 assignment document, given an array, a[i], ..., a[j], with $j - i \geq 2$, let $k = \lfloor (i + j)/2 \rfloor$, median-of-three partition process choose the median value among a[i], a[j], and a[k] (i.e., the middle value if a[i], a[j], and a[k] were sorted) as the partition element for QUICKSORT(A,p,r).

Pseudocode of Partition using Median of Three (Question a)

```
    MEDIAN(A, p, k, r):
1       if A[k] <= A[r]:
2           if A[p] <= A[k]:
3               return k
4           else:
5               if A[r] <= A[p]:
6                   return r
7               else:
8                   return p
9       else:
10          if A[k] <= A[p]:
11              return k
12          else:
13              if A[r] <= A[p]:
14                  return p
15              else:
16                  return r
```

```
    PARTITION_M3(A,p,r)
1       pivot=MEDIAN(A,i,j,k)
2       exchange A[pivot] with A[r]
3       x=A[r]
4       i=p-1
5       for j=p to r-1
6           if A[j]<=x
7               i=i+1
8               exchange A[j] with A[i]
9       exchange A[i+1] with A[r]
10      return i+1
```

```
    QUICKSORT_M3(A,p,r)
1       if p<r
2       q=PARTITION_M3(A,p,r)
3       QUICKSORT_M3(A,p,q)
4       QUICKSORT_M3(A,q+1,r)
```

Analysis of Worst Case Asymptotic Behavior of Quicksort using Median-of-Three Partition (Question b)

If duplicated values are allowed, the worst case of an input of size n is an array of n elements of the same value. In this case, each PARTITION_M3 process produces two subproblems: one subproblem of size n-1 and the other of size 0. In this case, each partition is extremely unbalanced and the time complexity of QUICKSORT_M3 is $\Theta(n^2)$. The analysis is totally the same with CLRS, page 193, worst-case analysis for QUICKSORT. To save space, we don't repeat the analysis here.

if duplicated values are not allowed, the analysis is as follows.

Assume that $T(n)$ is the worst case time for process QUICKSORT_M3 on an input of size n. Because the PARTITION_M3 produces two subproblems with total size $n-1$, we assume that the size of the first subproblem is q and the size of the second subproblem is $n-1-q$.

Considering that PARTITION_M3 is applied, the first subproblem must contain the smallest number of the three input numbers used for partition and the second subproblem must contain the biggest number of the three input numbers used for partition. So we have the following restriction for q:

$$q \in [1, n-2]$$

With the illustration of CLRS, page 193, worst-case analysis for QUICKSORT, we have the following recurrence for $T(n)$:

$$T(n) = \max(T(q) + T(n-1-q)) + \Theta(n)$$

a. Proof of $T(n) = O(n^2)$

Here we apply the substitution method and make the following assumption:

$$T(n) \le cn^2$$

Then we have:

$$T(n) = \max(T(q) + T(n-1-q)) + \Theta(n) \le \max(cq^2 + c(n-1-q)^2) + \Theta(n)$$

Assume that $f(q) = cq^2 + c(n-1-q)^2$, we have:

$$T(n) = \max(f(q)) + \Theta(n)$$

For $f(q)$, we have the following:

$$f(q) = cq^2 + c(n-1-q)^2 = c(q^2 + (n-1-q)^2) = c(2q^2 - 2(n-1)q + (n-1)^2)$$

So it is clear that $f(q)$ is a quadratic function whose symmetry axis is $x = (n-1)/2$. Considering that $q \in [1, n-2]$, we have:

$$\max(f(q)) = f(1) = f(n-2) \le f(0) = c(n^2 - 2n + 1)$$

So for $T(n)$, we have:

$$T(n) = \max(f(q)) + \Theta(n) \le c(n^2 - 2n + 1) + \text{Theta}(n) = cn^2 - 2cn + c + \Theta(n)$$

By picking the constant c large enough that $-2cn$ term dominates $\Theta(n)$ term, we have:

$$T(n) \leq cn^2 - 2cn + c + \Theta(n) \leq cn^2$$

So we get back to assumption made by the substitution method, which means that our substitution method is successful. So we have:

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

b. Proof of $T(n) = \Omega(n^2)$

Here we apply the substitution method and make the following assumption:

$$T(n) \geq cn^2 + dn$$

Then we have:

$$\begin{aligned} T(n) &= \max(T(q) + T(n-1-q)) + \Theta(n) \\ &\geq \max(cq^2 + dq + c(n-1-q)^2 + d(n-1-q)) + \Theta(n) \end{aligned}$$

Assume that $f(q) = cq^2 + dq + c(n-1-q)^2 + d(n-1-q)$, we have:

$$T(n) = \max(f(q)) + \Theta(n)$$

For $f(q)$, we have the following:

$$\begin{aligned} f(q) &= cq^2 + dq + c(n-1-q)^2 + d(n-1-q) \\ &= 2cq^2 - 2c(n-1)q + c(n-1)^2 + d(n-1) \end{aligned}$$

So it is clear that $f(q)$ is a quadratic function whose symmetry axis is $x = (n-1)/2$. Considering that $q \in [1, n-2]$, we have:

$$\max(f(q)) = f(1) = f(n-2) > f(2) = cn^2 + dn - 6cn + (13c - d)$$

So for $T(n)$, we have:

$$T(n) = \max(f(q)) + \Theta(n) > cn^2 + dn - 6cn + (13c - d) + \Theta(n)$$

By picking the constant c small enough that $\Theta(n)$ term dominates $-6cn$ term, we have:

$$T(n) > cn^2 + dn - 6cn + (13c - d) + \Theta(n) \geq cn^2 + dn$$

So we get back to assumption made by the substitution method, which means that our substitution method is successful. So we have:

$$T(n) = \Omega(n^2)$$

c. Proof of $T(n) = \Theta(n^2)$

Based on $T(n) = \Omega(n^2)$ and $T(n) = O(n^2)$, we have $T(n) = \Theta(n^2)$

Analysis of Worst Case Asymptotic Behavior of Quicksort using Median-of-Three Partition when Input Array is Sorted (Question c)

If duplicated values are allowed, the worst case of an input of size n is an array of n elements of the same value. In this case, each PARTITION_M3 process produces two subproblems: one subproblem of size n-1 and the other of size 0. In this case, each partition is extremely unbalanced and the time complexity of QUICKSORT_M3 is $\Theta(n^2)$. The analysis is totally the same with CLRS, page 193, worst-case analysis for QUICKSORT. To save space, we don't repeat the analysis here.

if duplicated values are not allowed, the analysis is as follows.

To fully utilize the fact that the input array is already sorted, we need to use the stable version of median-of-three partition. Namely, we need the partition to produce two subproblems while making sure that both the input for first subproblem and the input for second subproblem is already sorted. Here is the stable version of median-of-three partition:

```
    PARTITION_M3_STABLE(A,p,r)
1       pivot=MEDIAN(A,i,j,k)
2       x=A[pivot]
3       i=p-1
4       initialize a new array Smaller of length r-p
5       initialize a new array Bigger of length r-p
6       index_smaller=0
7       index_bigger=0
8       for j=p to r
9           if A[j]<x
10              index_smaller+=1
11              Smaller[index_smaller]=A[j]
12          else if A[j]>x
13              index_bigger+=1
14              Bigger[index_bigger]=A[j]
15      for j=p to p+(index_smaller-1)
16          A[j]=Smaller[j-(p-1)]
17      A[p+index_smaller]=x
18      for j=p+index_smaller+1 to r
19          A[j]=Bigger[j-(p+index_smaller)]
20      return p+index_smaller
```

Assume that T(n) is the worst case time for process QUICKSORT_M3 on an sorted input of size n.

Consider that when median-of-three partition is applied on a sorted input, the pivot is not only the middle value of the three values used for partition but also the middle value of the whole input. Considering that the pivot is element $\lfloor\frac{n+1}{2}\rfloor$, we have:

The size of the first subproblem is:

$$\lfloor\frac{n+1}{2}\rfloor - 1 = \lfloor\frac{n+1}{2} - 1\rfloor = \lfloor\frac{n-1}{2}\rfloor$$

(Note that $\lfloor x\rfloor + y = \lfloor x + y\rfloor \; if \; y \; is \; an \; integer$)

The size of the second subproblem is:

$$(n-1) - \lfloor\frac{n-1}{2}\rfloor = \lceil\frac{n-1}{2}\rceil$$

(Note that $x - \lfloor\frac{x}{2}\rfloor = \lceil\frac{x}{2}\rceil \; if \; x \; is \; an \; integer$)

We have the following recurrence for T(n):

$$T(n) = T(\lfloor\frac{n-1}{2}\rfloor) + T(\lceil\frac{n-1}{2}\rceil) + \Theta(n)$$

Considering that both $\lfloor\frac{n-1}{2}\rfloor$ and $\lceil\frac{n-1}{2}\rceil$ are very close to $\frac{n}{2}$, we omit the impact of some small reminder and floor or ceiling function. Then we have:

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

By case 2 of the master theorem, we have:

$$T(n) = \Theta(n \log n)$$

Implementation of Quicksort using the Normal Partition (Question d(i))

main.py implements the quicksort. To see more details, please refer to function partition and quicksort within this file.

Implementation of Quicksort using the Median of the Three Partition (Question d(ii))

File main.py implements the quicksort using median-of-three partition. To see more details, please refer to function partitionM3 and quicksortM3 within this file.

Test Run to Verify the Asymptotic Behavior of the Implementation of Quicksort using the normal partition (Question e(i)/f(iii))

Within PARTITION process, except the comparisons in the for loop, the time complexity is a constant. Assume that this constant is a1

Within QUICKSORT process, except the call for PARTITION process and the recursive call for QUICKSORT process itself, the time complexity is a constant. Assume that this constant is a2.

The time complexity of each comparison in “for” loop of PARTITION process is a constant. Assume that this constant is a3.

Then the total time complexity is:

$$T(n) = a1 * n1 + a2 * n2 + a3 * n3$$

,where $n1, n2, n3$ are the number of calls for PARTITION, calls for QUICKSORT and comparisons within the for loop of partition process.

To measure the asymptotic behavior, we need to measure $n1, n2, n3$.

File TestRunForQuickSort_Dataset1 records the input dataset.

File TestRunForQuickSort_Dataset2 records the output dataset.

File TestRunForQuickSort_Output records the asymptotic behavior.

For each n, it record $n1, n2, n3, n-1, 2n-1, n(n-1)/2$

For example, when $n = 11$, the test run shows that:

$n, n1, n2, n3, n-1, 2n-1, n(n-1)/2 = 11, 10, 21, 55, 10, 21, 55.0$ respectively.

Our test run shows us that:

$$n1 = n - 1$$

$$n2 = 2n - 1$$

$$n3 = n(n-1)/2$$

Consider that $T(n) = a1 * n1 + a2 * n2 + a3 * n3$, it is clear that $T(n) = \Theta(n^2)$

So our test run verifies the asymptotic behavior of the Implementation of quicksort using the normal partition.

Additionally, **WorstCaseBuilder.py** provides the method to build the worst case for quicksort using the normal partition. To build an example of the worst case for quicksort using the normal partition is relatively easy. Any array that is already sorted is an example. For example, input array [0,1,2,3,4,5] is an example of worst case when n=5.

Test Run to Verify the Asymptotic Behavior of the Implementation of Quicksort using the median of three partition (Question e(ii)/f(iii))

Within PARTITION_M3 process, except the comparisons in the for loop, the time complexity is a constant. Assume that this constant is a1

Within QUICKSORT_M3 process, except the call for PARTITION process and the recursive call for QUICKSORT process itself, the time complexity is a constant. Assume that this constant is a2.

The time complexity of each comparison in "for" loop of PARTITION process is a constant. Assume that this constant is a3.

Then the total time complexity is:

$$T(n) = a1 * n1 + a2 * n2 + a3 * n3$$

,where $n1, n2, n3$ are the number of calls for PARTITION, calls for QUICKSORT_M3 and comparisons within the for loop of partition process.

To measure the asymptotic behavior, we need to measure $n1, n2, n3$.

File TestRunForQuickSortM3_Dataset1 records the input dataset.

File TestRunForQuickSortM3_Dataset2 records the output dataset.

File TestRunForQuickSortM3_Output records the asymptotic behavior.

For each n, it record $n1, n2, n3, n/2, n, n^2/5$

For example, when $n = 11$, the test run shows that:

$n, n1, n2, n3, n - 1, 2n - 1, n(n-1)/2 = 11, 5, 11, 28, 5.5, 11, 24.2$ respectively.

Our test run shows us that:

$$n1 \approx n/2$$

$$n2 \approx n$$

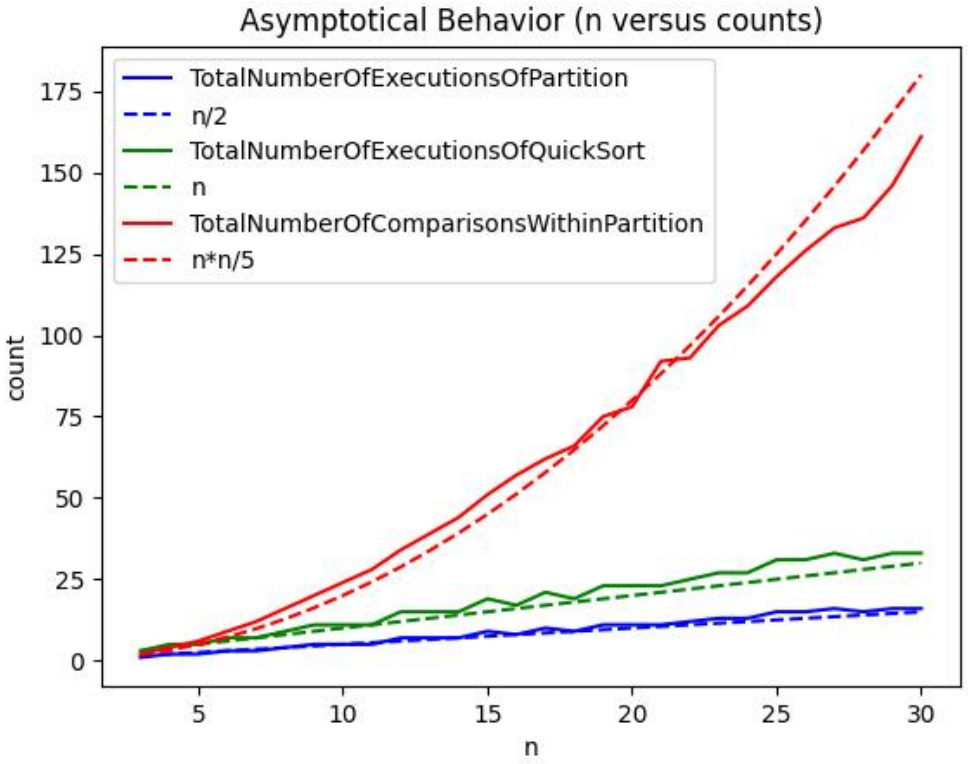
$$n3 \approx n^2/5$$

Consider that $T(n) = a1 * n1 + a2 * n2 + a3 * n3$, it is clear that $T(n) = \Theta(n^2)$

So our test run verifies the asymptotic behavior of the Implementation of quicksort using the median-of-three partition.

Additionally, **WorstCaseBuilder.py** provides the method to build the worst case for quicksort using median of three partition. To build an example of the worst case for quicksort using median of three is relatively hard. The idea of simulation is applied to build an example of this kind. Namely, we simulate running quicksort using median-of-three partition and revise the elements during the simulation. For example, [0, 1, 2.11, 3, 0.1, 2.1, 6, 3.1, 9.1, 9.11] is an example of worst case when n=10. To see more about this example, please refer to file **WorstCaseBuilder_Example**.

Comparison between the Algorithm's Worst-Case Asymptotic Behavior and the Program's Worst-Case Asymptotic Behavior (Question (g))



ComparisonForQuickSortM3.py draws the graph to compare algorithm's worst-case asymptotic behavior to the achieved asymptotic behavior of the program

ComparisonForQuickSortM3_Output is the output of ComparisonForQuickSortM3.py, comparing algorithm's worst-case asymptotic behavior to the achieved asymptotic behavior of the program

In this graph, we can see that when the input scale n ranges from 3 to 30, we have the following conclusions:

1. The number of executions of partition process is very close to $n/2$
2. The number of executions of quicksort process is very close to n
3. The number of comparisons in partition process is very close to $n^2/5$

These conclusions drawn from this graph verifies the worst-case asymptotic behavior of the Implementation of quicksort using the median-of-three partition. Namely, $T(n) = \Theta(n^2)$