Hunter Caskey, (716) 720-2564
Professor John Boon
Foundations of Algorithms
JHU Whiting School of Engineering
21 October 2019

Programming Assignment #2

Given an array , a[i], ..., a[j], with $j - i \geq 2$, let k = b(i + j)/2c, choose the median value amongst 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). This is called median-of-three partitioning.

1a. [5 points] Write pseudocode for PARTITION(A.p.r), page 171, using median-of-three partitioning.

```
MEDIAN-OF-THREE(A, p, q, r):
if $A[p] \leq A[q]$ and $A[q] \leq A[r]$ or $A[r] \leq A[q]$ and $A[q] \leq A[p]$
    return q
else if $A[q] \leq A[p]$ and $A[p] \leq A[r]$ or $A[r] \leq A[p]$ and $A[p] \leq A[q]$
    return $p$
else
    return $r$

$PARTITION(A, p, r)$:
$q = \lfloor (p+r)/2 \rfloor$
$pivot =$ MEDIAN-OF-THREE(A, p, q, r)
$x = A[pivot]$
$i = p - 1$
for $j = p$ to $r - 1$
    if $A[j] \leq x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i+1]$ with $A[r]$
return $i + 1$
```

1b. [5 points] What is the running time of PARTITION(A.p.r) using median-of-three partitioning? Provide a proof supporting your conclusion.

The running time of PARTITION(A, p, r) using median-of-three partitioning has the same asymptotic complexity as the PARTITION algorithm from the text on Page 171. The only significant difference between the text's version of PARTITION and the one utilizing median-of-three is the calculator of the middle element of the subarray A and the selection of the pivot step. The selection of the pivot, expressed in the routine MEDIAN-OF-THREE, is simply a three-branch conditional statement that executes in a runtime on the order of O(1). Since the only portion of the algorithm that changed was how the pivot is selected, and since the selection of the pivot is still done in constant time, it follows that no additional cost occurs in the for-loop which acts as the dominant factor for the complexity of the algorithm. Hence, the PARTITION routine using median-of-three partitioning has a runtime of O(n).

1c. [10 points] What is the running time of QUICKSORT(A.p.r) using median-of-three partitioning on an input set that is already sorted? Provide a proof supporting your conclusion.

Let n be the size of the initial input array, A. If QUICKSORT using median-of-three partitioning is called with an input array that is already sorted, then this would result in the best-case performance for the algorithm. In this case, since the input is sorted, everytime that PARTITION subroutine is called, a true median of the subarray would be returned and used as the pivot. Assuming an even parity input set, this would consequently result in the division of the initial array into two subarrays of equal size. Each recursive call to QUICKSORT would then result in processing of half of the input size. Due to this 'halving' effect of the input on successive calls to QUICKSORT, it follows that there can be at most $log_2 n$ recursive calls before a trivial case of a subarray of size less than or equal to 1 occurs. Furthermore, since each of the invocations of QUICKSORT at each level of the 'call tree' result in a distinct portion of the original input array being processed, it follows that on each level exactly n elements are processed. Thus, the complexity of each individual level of the call tree is on the order of $O(n)$, and when paired with the overall depth of this tree, the complexity of the overall QUICKSORT algorithm becomes $O(n\ log\ n)$.

1d. [20 points] Implement QUICKSORT(A.p.r), using:

i. [10 points] PARTITION(A.p.r) on page 171, and

ii. [10 points] PARTITION(A.p.r) using median-of-three partitioning.

For parts d.i and d.ii above, please reference the accompanying source code files and supporting material delivered with this assignment.

1e. [50 points] Test your implementations to verify analytically derived complexity bounds:

i. [20 points] Average and worst case runtimes for (d.i.) implementation of QUICKSORT(A.p.r) with PARTITION(A.p.r) on page 171, and

See results of testing in quicksort_tests.log.

ii. [20 points] Average and worst case runtimes for (d.ii.) implementation of QUICKSORT(A.p.r) with PARTITION(A.p.r) using median-of-three partitioning.

See results of testing in quicksort_tests.log.

iii. [10 points] Analysis of the results obtained by testing your implementations.

Based off of the results described in quicksort_tests.log, the median-of-three changes made to the quicksort algorithms drastically improved performance of the sort in certain circumstances. The enhanced version of quicksort seemed to show off when the input set was already sorted or mostly sorted, and by my observations there was about a 25% difference in the timing metrics for each run of the algorithm. Other test cases of the input being in complete reverse order or being for the most part randomized seemed to be inconclusive in terms of an increased benefit on performance. I believe that this can be attributed to the overall increase in operations that were used to construct the median-of-three quicksort compared to the naive approach.

1f. [5 points] Is pseudocode consistent with text style requirements?

See pseudocode above.

1g. [5 points] Are metrics used to analyze running code consistent with the Programming Assignment Guidelines

See results of testing in quicksort_tests.log.