Name: Guoqing Yu

Problem #2

1. *A* = [17, 10, 25, 35, 29, 15, 22]

* To start with, we implement partitioning on subarray *A*[1…7] (*p* = 1, *r* = 7):
  + Pick the last element *A*[7] as the pivot: *x* = 22
  + Set *i* as 0 and traverse *j* from 1 to 6:
  + *i* = 0, *j* = 1 => *A*[1] = 17 < 22 => *i* increments to 1, exchange *A*[1] with *A*[1] yields no change.
  + *i* = 1, *j* = 2 => *A*[2] = 10 < 22 => *i* increments to 2, exchange *A*[2] with *A*[2] yields no change.
  + *i* = 2, *j* = 3 => *A*[3] = 25 > 22
  + *i* = 2, *j* = 4 => *A*[4] = 35 > 22
  + *i* = 2, *j* = 5 => *A*[5] = 29 > 22
  + *i* = 2, *j* = 6 => *A*[6] = 15 < 22 => *i* increments to 3, exchange *A*[3] with *A*[6] to get: [17, 10, 15, 35, 29, 25, 22]
  + Swap *A*[4] with *A*[7] to get: [17, 10, 15, 22, 29, 25, 35]
  + Return 4.
* Partition on *A*[1…3] (*p* = 1, *r* = 3):
  + Pick the last element *A*[3] as the pivot: *x* = 15
  + Set *i* as 0 and traverse *j* from 1 to 2:
  + *i* = 0, *j* = 1 => *A*[1] = 17 > 15
  + *i* = 0, *j* = 2 => *A*[2] = 10 < 15 => *i* increments to 1, exchange *A*[1] with *A*[2] to get: [10, 17, 15, 22, 29, 25, 35]
  + Swap *A*[2] with *A*[3] to get: [10, 15, 17, 22, 29, 25, 35]
  + Return 2
* Partition on *A*[1…1] is base case.
* Partition on *A*[3…3] is also base case.
* Partition on *A*[5…7] (*p* = 5, *r* = 7):
  + Pick the last element *A*[7] as the pivot: *x* = 35
  + Set *i* as 4 and traverse *j* from 5 to 6:
  + *i* = 4, *j* = 5 => *A*[5] = 29 < 35 => *i* increments to 5, exchange *A*[5] with *A*[5] yields no change
  + *i* = 5, *j* = 6 => *A*[6] = 25 < 35 => *i* increments to 6, exchange *A*[6] with *A*[6] yields no change
  + Swap *A*[7] with *A*[7] yields no change.
* Partition on *A*[5…6] (*p* = 5, *r* = 6):
  + Pick the last element *A*[6] as the pivot: *x* = 25
  + *i* = 4, *j* = 5 => *A*[5] = 29 > 25
  + Swap *A*[5] and *A*[6] to get: [10, 15, 17, 22, 25, 29, 35]

1. *A* = [10, 15, 20, 25, 30, 35, 40]

In the first partition of the entire array of length 7, we need *n* = 6 total comparisons. Since all elements on the left side of the pivot are smaller than x, none of elements are swapped (*i* and *j* always increment concurrently and swapping them yields no difference). The returned value of the first partition is 7.

Then, we move on to partition the subarray *A*[1…6]. Similarly, we need 5 comparisons for a subarray of length 6, with a returned value of 6 because none of elements are swapped.

Then, we move on to partition the subarray *A*[1…5], which we need 4 comparisons.

…

Finally, we partition the subarray *A*[1…2], where only 1 comparison is needed.

As a result, we need 6 + 5 + 4 + … + 1 = 21 total comparisons for quicksort on array *A*.

1. In the first partition of the entire array of length *n*, we need *n* – 1 total comparisons.

Suppose the returned value is *q*. Then, we need to partition the subarrays *A*[1…*q* – 1] and *A*[*q* + 1…*n*] respectively.

*A*[1…*q* – 1] requires *q* – 2 comparisons if *q* – 1 ≥ 1 (*q* ≥ 2), or no comparison if *q* < 2.

[*q* + 1…n] requires *n* – *q* – 1 comparisons if *q* + 1 ≤ *n* (*q* ≤ *n* – 1), or no comparison if *q* > *n* – 1.

Therefore, if *q* < 2, there should be *n* – *q* – 1 comparisons for the two subarrays.

If 2 ≤ *q* ≤ *n* – 1, there should be (*q* – 2) + (*n* – *q* – 1) = *n* – 3 comparisons.

If *q* > *n* – 1, there should be *q* – 2 comparisons.

Clearly, *n* – *q* – 1 > *n* – 3 if *q* < 2. *q* – 2 > *n* – 3 if *q* > *n* – 1.

To sum up, a returned value *q* within [2, *n* – 1] would offer the least number of comparisons (i.e., *n* – 3) for the next round.

So, for an array of length 7, the first partition should conduct 6 comparisons and return a value within [2, 6] to minimize the number of future comparisons.

*A*[1…*q* – 1] and *A*[*q* + 1…7] should have 7 – 3 = 4 comparisons in total.

For partitioning *A*[1…*q* – 1], we expect a returned value within [2, *q* – 2] (thus, *q* ≥ 4) to gain the least number of comparisons (i.e., *q* – 4) for next level.

For partitioning *A*[*q* + 1…7], we expect a returned value within [*q* + 2, 6] (thus, *q* ≤ 4) to gain the least number of comparisons (i.e., 4 – q) for next level.

Therefore, picking *q* = 4 would achieve the minimum number of comparisons, which is 6 + 4 + (*q* – 4) + (4 – *q*) = 10.

The exact process is: First, partitioning on *A* requires 6 comparisons and returns 4, which divides *A* into *A*[1…3] and *A*[5…7]. Then, partitioning on *A*[1…3] requires 2 comparisons and returns 2. Partitioning on *A*[5…7] also requires 2 comparisons and returns 6. No more comparison is needed. The minimum number of total comparisons is 10.

1. To get the minimum number of total comparisons:

Since *n* = 2*k* – 1, the first partition on the entire array would require 2*k* – 2 comparisons.

As shown in (c), the optimal way for the next level partitioning would be to choose the middle point of the array as the return value.

Therefore, the array is further divided into two subarrays of length .

Similarly, each of these two subarrays require 2k-1 – 2 comparisons, which sums upto 2(2*k*-1 – 2) for Round 2.

These two subarrays are further divided into 4 subarrays of length in Round 3.

…

We can conclude that the minimum number of total comparisons is equal to:

.

For all

So .

Therefore, we can pick *c* = 2 and *n*0 = 1 to show that the running time of Quicksort is in the best case.

To get the maximum number of total comparisons:

Suppose the array is already sorted like (b), then in each round we always have to handle a subarray whose length is one element shorter than the parent subarray (e.g., [1, 2, 3, 4, 5] => [1, 2, 3, 4] => [1, 2, 3] …).

Since each round requires *n* – 1 comparisons (Here, *n* refers to the length of subarray), the maximum number of total comparisons is equal to:

.

For all

So .

Therefore, we can pick *c* = and *n*0 = 1 to show that the running time of Quicksort is in the worst case.