7.15 3,1,4,1,5,9,2,6

⇓

(3,1,4,1) (5,9,2,6)

⇓

(3,1), (4,1) (5,9), (2,6)

⇓

3,1,4,1,5,9,2,6

⇓

(1,3), (1,4), (5,9), (2,6)

⇓

(1,1,3,4) (2,5,6,9)

⇓

1<2
1,
1<2
1,1,
3>2
1,1,2
3<5
1,1,2,3,
4<5
1,1,2,3,4

⇓

1,1,2,3,4,5,6,9 Final sorted array.

7.17

a. Sorted input : O(nlogn) *Note if its Natural merge sort, it will be O(
b. reversed-order input: O(nlogn)
c. random input : O(nlogn)

7.19 3,1,4,1,5,9,2,6,5,3,5

① find median from beginning, middle and end. 3, 9, 5 ; median = 5

move 9 and 5

Index: 0 1 2 3 4 5 6 7 8 9 10

3, 1, 4, 1, 5, 9, [5], 2, 6, 5, 3, 9 Pivot = 5

P

② swap pivot with second-last

Index: 0 1 2 3 4 5 6 7 8 9 10

3, 1, 4, 1, 5, 3, 2, 6, 5, [5], 9

Set i=0, j=9 where pivot is.

when i=6, j=8, swap the values

6>5 i=6 j=8

3, 1, 4, 1, 5, 3, 2, 5, 6, [5], 9

when i=8, j=7, crossed, swap i with pivot.

i j

Index 0 1 2 3 4 5 6 7 8 9 10

3, 1, 4, 1, 5, 3, 2, 5, [5], 6, 9

Take left side's subarray.

previous pivot

0 1 2 3 4 5 6

3, 1, 4, 1, 5, 3, 2, 5

median = 1, 3, 5

Pivot = 3

swap with 2nd last element

1, 1, 4, [3], 5, 3, 2, 5

i=0 j=6

when i=2 j=5

1, 1, 4, 2, 5, 3, [3], 5

i j

1, 1, 3, 2, 5, 4, 6, 5

i j

0 1 2 3 4 5 6 7

1, 1, 3, 2, 5, 4, [3], 5

i j

5, 6, 9 4

when i = 4, j = 3,

swap i with pivot

1, 1, 3, 2, 3, 4, 5, 5

new subarray = 1, 1, 3, 2

median = 1,

1, 1, 3, 2 → 1, 3, [1], 2

i j

swap 1 with 2nd last

when i = 1, j = 0

swap i with pivot

1, [1], 3, 2

③ 1, [1], 3, 2, [3], 4, 5, 5, [5], 6, 9

Use insertion sort index 2 and 3, "3, 2" → "2, 3"

since cutoff = 3

④ final array: 1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9

7.20

a. sorted input: $O(N \log N)$

b. reversed input: $O(N \log N)$

c. random input: $O(N \log N)$

7.21

a. the first element: $O(N^2)$

b. larger of the first two distinct elements:

c. a random element: $O(N^2)$

d. the average: $O(N \log N)$

7.22

a. The worst case will be $O(N \log N)$ as the implementation uses median 3- partitioning

b. the two while loops need to be changed to

line 19 ① while (a[++i] <= pivot && i<j)

line 20 ② while (pivot >= a[--j] && j>i)

The run time will be $O(N^2)$ since the median selected will be the same always, so it is worst-case scenario.

c.

~~① while (a[++i] <= pivot)~~

line 20 ① while (pivot <= a[--j] && j>i)

The run time will be $O(N^2)$ when keys are equal

7.23.

Unless the array is sorted, picking the middle can still be the smallest/largest value causing it to have $O(N^2)$ time. But if it is sorted, it will greatly reduce the chance that quicksort will require quadratic time.

7.24

Let The array be [20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

This median-3 will always be 20, which will be worst case scenario.