**1. (CLRS 6.2-6)**

Suppose the height of the heap is .

At each level, it takes exactly 2 comparisons and 1 swapping.

The worst case is a leaf node is larger than the root, witch need to be operated all the way up to the root. So it takes h iterations and the total running time is .

Since heap is an almost complete binary tree, equals to . Then the worst case of MAX-HEAPIFY is .

**2. (CLRS 6.4-3)**

If array A is built in an increasing order: first do BUILD-MAX-HEAP, takes , and then iteratively swap the root to end and MAX-HEAPIFY the rest, totally taking . So the total running time is . This is the worst case.

If array A is built in a decreasing order: the first step can be skipped, then the total running time would simply be . This is the best case.

**4. (CLRS 6-2)**

**a.** From up to down levels, put elements in the array from left to right on the same level. Left-most child of node is , the child is ; parent of node is

**b.**

**c.** EXTRACT-MAX

1 exchange the root element with the last

2 decrease the size of heap by 1

3 call MAX-HEAPIFY on the new root, on a heap of size

The first two steps takes exactly 1 operation each. MAX-HEAPIFY contains d comparisons at each level and the height of heap is , so total cost of one MAX-HEAPIFY call is .

In total, the running time of EXTRACT-MAX of d-ary heap is .

**d.** INSERT(A, key, n)

1 increase the size of heap by 1

2

3 call HEAP-INCREASE-KEY(A,n+1,key)

Running time of HEAP-INCREASE-KEY is , other operations are simply constants, so the total running time of INSERT is .

**e.** INCREASE-KEY(A, i, k)

1 if k < A[i]

2 error “new key is smaller than current key”

3 A[i]=k

4 while i>1 and A[Parent(i)] < A[i]

5 exchange A[i] and A[Parent(i)]

6 i=parent(i)=

As discussed in (d), total running time of INCREASE-KEY is .

**6. (CLRS 7.2-3)**

If array A is sorted in decreasing order, the first(largest) or last(smallest) element in the array is often chosen to be partitioning pivot. So in every iteration of partitioning, the array is partitioned into 1 and (n-1) length, i.e. the operation size only reduce by one at each loop. Then it takes n-1 times iteration of partition, and in every iteration it takes k comparisons where k is the current array size. Thus the total running time is .

**7. (CLRS 7-2)**

**a.** If all elements are equal, it takes only one iteration of partition and then all elements are checked on the correct position. So the total running time is .

**b.** PARTITION’(A, p, r)

1 x=A[r]

2 i=p-1

3 k=p-1

4 for j=p to r-1

5 if A[j]<x

6 i=i+1

7 k=k+1

7 exchange A[i] with A[j]

8 else if A[j] = x

9 k=k+1

10 Exchange A[k] with A[j]

11 exchange A[k+1] with A[r]

12 return i+1, k

**c.**

RANDOMIZED-PARTITION’ (A, p, r)

1 i = RANDOM(p, r)

2 exchange A[r] and A[i]

3 return PARTITION’ (A, p, r)

RANDOMIZED-QUICKSORT’ (A, p, r)

1 if p<r

2 q, t = RANDOMIZED-PARTITION’ (A, p, r)

3 RANDOMIZED-QUICKSORT’ (A, p, q-1)

4 RANDOMIZED-QUICKSORT’ (A, t+1, r)

QUICKSORT’ (A, p, r)

1 return RANDOMIZED-QUICKSORT’ (A, p, r)

**d.** “All elements are distinct in array A” is no longer an assumption but a worst case in the PARTITION’ algorithm. So the upper-bound is .

**8.**

A

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 3 | 4 | 5 | 6 | 7 | 13 | 19 | 15 | 16 | 11 |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 9 | 9 | 10 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  | 11 |  |  |  |  |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 6 | 7 | 7 | 8 | 9 | 9 | 9 | 10 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  | 11 |  |  | 16 |  |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 6 | 7 | 7 | 8 | 8 | 9 | 9 | 10 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  | 11 |  | 15 | 16 |  |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 6 | 7 | 7 | 7 | 8 | 9 | 9 | 10 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  | 11 |  | 15 | 16 |  |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 6 | 7 | 7 | 7 | 8 | 9 | 9 | 9 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  | 11 | 13 | 15 | 16 | 19 |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 9 | 9 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  | 7 | 11 | 13 | 15 | 16 | 19 |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 5 | 5 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 9 | 9 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  | 6 | 7 | 11 | 13 | 15 | 16 | 19 |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 9 | 9 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  | 5 | 6 | 7 | 11 | 13 | 15 | 16 | 19 |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 9 | 9 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 4 | 5 | 6 | 7 | 11 | 13 | 15 | 16 | 19 |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 1 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 9 | 9 |

B

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 3 | 4 | 5 | 6 | 7 | 11 | 13 | 15 | 16 | 19 |

C

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 0 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 9 | 9 | 9 |

**9. (CLRS 8.3-3)**

We need to prove that the radix algorithm works on a list of up to digit numbers. (If an element is less than digit long, we consider that its higher digits are 0.)

First, we will show it works when . This is no other than a simple sorting problem and if the algorithm is stable, we will get the correct sorted list.

Secondly, we assume that it works for , i.e. all of the numbers can be sorted correctly when they are digits long.

Now we prove it for We can use the former digits lists and randomly add a digit as their most-significant digit. Then we sort this new-added digits in place. Suppose we have the two in order numbers and , such that before adding the highest digit. Adding the highest digit and sorting them would either keep or exchange their position, while other numbers would not be touched, and then the final relative position between and are ensured to be correct. As for the other case, when the highest digit are equal, their position will remain unchanged, so they are still in right order based on their lower digits. Consequently all pairs of the digit long numbers are in relatively correct position, then we know that the whole list are sorted correctly.

In summary, the radix algorithm is proved to work right.

When sorting every digit, we need the sorting algorithm to be stable, which ensure the whole d-digit number to be sorted instead of just arranging the one digit.

**10. (CLRS 9-1)**

**a.** Merge Sort, Heap Sort or Quick Sort are eligible, and their worst-case running time are And list the largest takes , so in total it takes

**b.** Using heap data structure and do BUILD-MAX-HEAP takes . The EXTRACT-MAX operation takes each time, so calling times takes in total.

**c.** Finding the th largest number using SELECTION algorithm takes . Then the PARTITION algorithm is called once, which takes . And sorting elements takes in worst case. In total, this procedure takes .