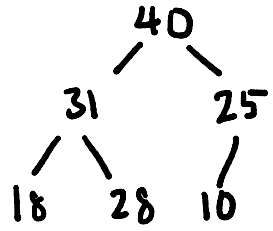
CS 240 F2007 Midterm

1. a) F b) F c) T d) T e) T f) F g) T h) F i) ??

2. a)



b) k will be located at index n+1. The initial step inserts k into the first free leaf, which is at index n+1. Since key k is smaller than all other keys, it is hence smaller than its parent, and is not fixed up, since the heap order property is maintained.

3. The initial pivot should be the element 9, since this will result in the array being split into exactly half.

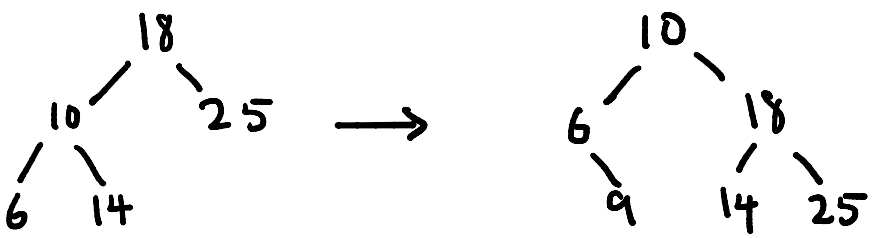
4. A) Radix sort should be used. Since the keys are all positive integers with the same number of digits, radix sort can be used to sort in O(n) time, where n is the number of students.

B) Radix sort should be used. Since all the keys are integers from 0 to 100, we know the max number of digits in each number is 3, which is constant, so we can sort in O(n) time, where n is the number of students. (can also use count sort since integer range is fixed)

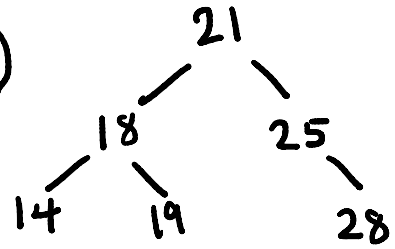
C) Mergesort can be used, since it is a comparison-based sorting algorithm, and runs in O(nlogn) time. (can also use randomized quicksort)

D) Insertion sort can be used. Since the data is mostly sorted, there will be few elements that need comparisons, which means insertion sort will be faster.

5. a)



b)



6. We didn't do this

7. Or this :P

8. a)

0
1
2 22
3 13 43
4 34

b)

0 74
1
2 52
3 83
4 44

c)

0
1 43
2 22
3 13
4 34

9. The for loop runs from 1 to n, so takes n time (assuming constant-time operations take 1 time). The last line takes T(n/2) time. T(1) = 1. Hence we have

$$T(n) = n + T(n/2)$$

We prove by induction that T(n) <= 2n. Our base case is T(1) = 1. We have that

$$T(n) = n + T(n/2) \leq n + 2\left(\frac{n}{2}\right) = n + n = 2n$$

We prove by induction that T(n) >= n. Our base case is T(1) = 1. We have that

$$T(n) = n + T(n/2) \geq n + \frac{n}{2} = \frac{3n}{2} \geq n$$

Hence we have that T(n) is in Theta(n).

10. Suppose we have an implementation of the Priority Queue ADT that guarantees an O(1) running time for both ExtractMin and Insert. Then for a list of n items, we can sort them by inserting into this priority queue n times, in O(n) time, then ExtractMin n times, in O(n) time, to sort the array. This would sort the array in O(n) time, which contradicts the theorem that any comparison-based sorting algorithm must use Omega(nlogn) time.