Name: _____

1. (3 points) What is the running time of QUICKSORT in each of the following cases?

(a) Input A contains distinct elements sorted in decreasing order	$\Theta(n^2)$
(b) Input A contains distinct elements sorted in increasing order	$\Theta(n^2)$
(c) Input A results in constant-proportion partitions	$\Theta(n \lg n)$

2. (1 point) Justify your answer to (1.b).

Solution

Since the elements are distinct and in ascending order, the pivot, $A[r]$, is larger than every other element of A. Therefore, the conditional on line 4 of PARTITION is always true and i will be incremented with every iteration of the loop, resulting in $i == j$ at every iteration, and so the exchange on line 6 does nothing. The loop terminates with $i = r - 1$, so the final exchange on line 7 also does nothing, leaving the pivot in its original position, and returning r to QUICKSORT as the "midpoint" of the array. QUICKSORT will therefore make a *single* recursive call with start and end indices p and $r - 1$; that is, the recursive call will be one element smaller, which is maximally imbalanced. We have shown that QUICKSORT with maximally imbalanced partitions has a quadratic running time.

(continued on other side)

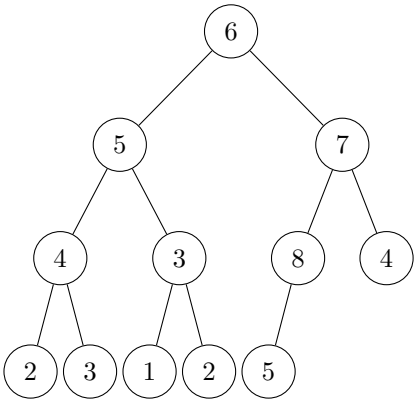
3. (3 points) Demonstrate the execution of MAX-HEAPIFY$(A, 1)$ on the heap $A = \langle 6, 5, 7, 4, 3, 8, 4, 2, 3, 1, 2, 5 \rangle$.

(a) Draw the tree representation of the heap before the call to MAX-HEAPIFY.

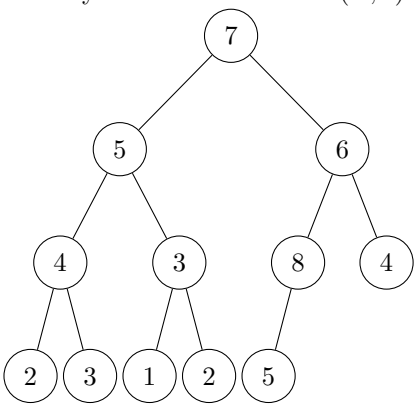
(b) Describe each step of the algorithm and show how the tree changes.

(c) Write the final state of the array A.

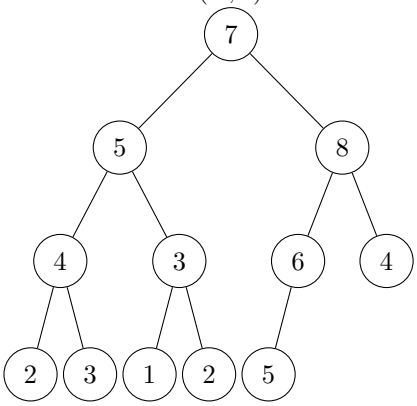
Solution



MAX-HEAPIFY compares nodes 1, 2, and 3; determines that nodes 1 and 3 (values 6 and 7) need to be swapped. Recursively calls MAX-HEAPIFY$(A, 3)$.



Call toMAX-HEAPIFY$(A, 3)$ determines that nodes 3 and 6 (values 6 and 8) need to be swapped. Recursively calls MAX-HEAPIFY$(A, 6)$.



Call to MAX-HEAPIFY$(A, 6)$ determines that no swap is needed. Done.

4. (3 points) Use induction on d, the number of digits in each array element, to prove that radix sort works.

Solution

If $d = 1$, then radix sort is just counting sort with $k = 9$, so it correctly sorts the numbers. Now, suppose radix sort works for $d - 1$ digits; we need to show that it works for d digits. The first $d - 1$ iterations will correctly sort the integers on their $d - 1$ low digits, by assumption. On the d^{th} iteration, counting sort will correctly sort on the high digit, grouping the integers with leading '0' together, followed by the integers with leading '1', etc., and since counting sort is stable, the integers will remain ordered on their low $d - 1$ digits within each group. Therefore the integers will be sorted correctly.

(pseudocode on other side)

QUICKSORT(A, p, r)

```
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

PARTITION(A, p, r)

```
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          Exchange A[i] and A[j]
7  Exchange A[i + 1] with A[r]
8  return i + 1
```

MAX-HEAPIFY(A, i)

```
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap_size and A[l] > A[i]
 4      largest = l
 5  else largest = i
 6  if r ≤ A.heap_size and A[r] > A[largest]
 7      largest = r
 8  if largest ≠ i
 9      Exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

RADIX-SORT(A, d)

```
1  for i = 1 to d
2      Sort A on digit i using a stable sort
```