CS 3DA3 Problem Set II Solutions
October 23, 2007

Question 1.

1.1 (First Solution)
Let's suppose that we wish to sort (a, b, c, d). There are six comparisons that we can consider. Among any set of 4 comparisons, there must be two involving all four elements, say (a, b) and (c, d) and one involving an element from each of these pairs, say (a, d). All three of these give the same results on the following inputs:

1. $(1, 2, 3, 4)$
2. $(1, 3, 2, 4)$
3. $(1, 4, 2, 3)$

There are three possibilities for the fourth comparison:

- (a, c) – gives the same result on (1) and (2).
- (b, c) – gives the same result on (2) and (3).
- (b, d) – gives the same result on (1) and (2).

No matter what set of 4 comparisons is chosen, there will be two distinct inputs that will be permuted in the same way and therefore one of them cannot be properly sorted.

1.1 (Second Solution)
There are $4! = 24$ permutations of $(1, 2, 3, 4)$. Let S be the set of such permutations. After one comparison, S is partitioned into two subsets according to the outcome of the comparison. At least one of these, call it S_1 has at least 12 elements.

Similarly, after two comparisons, S_1 is partitioned into two subsets, at least one of which, call it S_2, has 6 or more elements. Continuing, we get S_3 and S_4 with at least 3 and 2 elements respectively. Any four comparisons must agree on a set of 2 elements and therefore a fifth comparison is required.

1.2

The following is inspired by the **mergesort** algorithm. It outputs a, b, c, and d in sorted order. If you wish, the output statements can be replaced by assignment to 4 new variables or to an array. Note that the number of comparisons is either 4 or 5. The code for **swap** is trivial and is omitted.

```
if a > b
  swap(a,b)
endif
if c > d
  swap(c, d)
endif
if a < c
  output a
  Merge (b, c, d)
else
  output c
  Merge (d, a, b)
endif
```

where **Merge** is described as follows:

```
Merge(x, y, z)
Precondition: (y, z) already sorted
if x < y
  output x; output y; output z
else
  output y
  if x < z
    output x; output z
  else
    output z; output x
  endif
endif
endMerge
```

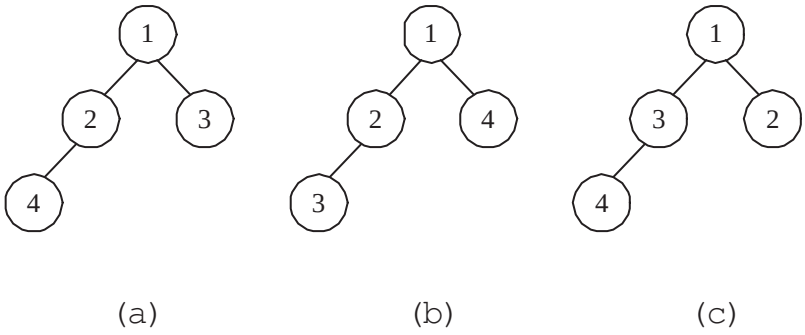


Figure 1: Question 1.4

1.3
The best answer is 7. Following the logic of the second solution to Problem 1, after k comparisons, you have 2^k distinct outcomes (at most) and you need to distinguish among $5! = 120$ permutations. Thus we need $2^k \geq 5!$ which means $k \geq 7$. This observations generalizes to sorting any n items. You need $2^k \geq n!$. This is a basis for showing that the worst case number of comparisons is in $\Omega(n \log n)$ which is a fundamental result on the complexity of sorting algorithms.

1.4
Assuming the heap order is: key(parent) $\leq$ key(children). Then there are three possible heaps:

- $(a)(1, 2, 3, 4)$
- $(b)(1, 2, 4, 3)$
- $(c)(1, 3, 2, 4)$

1.5
If we insert each element one by one then the following sequence will yield heap $(a), (b)$ and (c) respectively.

(a)
$(1,2,3,4),(1,4,3,2),(2,1,3,4),(2,4,3,1),(3,4,1,2),$
$(3,4,2,1),(4,1,3,2),(4,2,3,1),(4,3,1,2),(4,3,2,1)$

(b)
$(1,2,4,3),(1,3,4,2),(2,1,4,3),(2,3,4,1),(3,1,4,2),\ (3,2,4,1)$

(c)
$(1,3,2,4),(1,4,2,3),(2,3,1,4),(2,4,1,3),(3,1,2,4),$
$(3,2,1,4),(4,1,2,3),(4,2,1,3)$

Question 2.

2.1
If a bound N is known in advance on the total number of elements that will ever be in the heap at any time, then we can set up an array to store the heap. Heaps are maintained in an array H indexed by $i = 1, ..., N$. We will think of the heap nodes as corresponding to the positions in this array. $H[1]$ is the root, and for any node at position i, the children are the nodes at positions $\texttt{leftChild}(i) = 3i - 1$ and $\texttt{midChild}(i) = 3i$ and $\texttt{rightChild}(i) = 3i + 1$. If the heap has $n < N$ elements at some time, we will use the first n positions of the array to store the n heap elements and use $\texttt{length}(H) = 3i$ to denote the number of elements in H. This representation keeps the heap balanced at all times.

2.2
For a node whose position in the array is i, as described in previous question, its children are $3i-1, 3i, 3i+1$ respectively. This can be proved by induction.

Proof. It is sufficient to prove $\texttt{leftChild}(i) = 3i - 1$. Base case: the first nodes 1, 2, 3 and 4 satisfy the relation. Suppose for all $j < i$, the relation satisfies. Then $\texttt{leftChild}(i) = (3(i-1)+1)+1 = 3i-1$. □

Accordingly, the parent of a node is $\lfloor (i+1)/3 \rfloor$.

2.3

First observe that for a full heap, in each level l, the number of nodes in this level is 3^l (suppose the root is in level 0). Then the total number of nodes (also the index of the last element) of a full heap of level l is $\frac{3^{l+1}-1}{3-1}$. For an unfull heap with n nodes, suppose it has l levels, then the index of the last element in level $l-1$ is $\frac{3^l-1}{3-1}$. Also $\frac{3^l-1}{3-1} < n \leq \frac{3^{l+1}-1}{3-1} \Rightarrow \log_3(2n+1) - 1 \leq l < \log_3(2n+1) \Rightarrow l = \lceil \log_3(2n+1) - 1 \rceil$. So all nodes with index n', where $\frac{3^{l-1}-1}{3-1} < n' \leq n$, are leaf nodes.

2.4
If we define the height of a heap as the largest level l in the previous question, then as in the previous question, $l = \lceil \log_3(2n+1) - 1 \rceil$.

Question 3.

3.1
To implement *Insert* and *ExtractMin*, we first define two subroutines *Heapify-up* and *Heapify-down* mimic to the one in the text book as follows:

```
Heapify-up(H, i):
  If i>1 then
    let j=parent(i) = floor((i+1)/3)
    If key[H[i]]<key[H[j]] then
      swap the array entries H[i] and H[j]
      Heapify-up(H,j)
    Endif
  Endif

Heapify-down(H, i):
  Let n = length(H)
  If 3i-1>n then
     Terminate with H unchanged
  Else if 3i+1 =< n then
     Let left = 3i-1, mid = 3i and right = 3i+1
     Let j be the index that minimizes key[H[left]],
     key[H[mid]] and key[H[right]]
  Else if 3i == n then
     Let left = 3i-1 and mid = 3i
     Let j be the index that minimizes key[H[left]] and key[H[mid]]
```

```
  Else
     Let j = left
  Endif
  if key[H[j]] < key[H[i]] then
     swap the array entries H[i] and H[j]
     Heapify-down(H,j)
  Endif
```

Now the *Insert* and *ExtractMin* function is as follows.

```
Insert(H, v):
  newlength = length(H)+1
  H[newlength] = v
  Heapify-up(H, newlength)

ExtractMin(H):
  result = H[1]
  length = length(H)
  H[1] = H[length]
  Heapify-down(H, 1)
  return(result)
```

3.1

Proof. For a sequence of k packages indexed by $(1, 2, ..., k)$, suppose the current algorithm uses l trucks denoted by a sequence $(T_1, T_2, ..., T_l)$. Denote the last package in a truck by $(L_1, L_2, ..., L_l)$, then $L_1 < L_2 < L_3 ... < L_l$ and $(L_1, L_2, ..., L_l)$ is a subsequence of $(1, 2, ..., k)$. For example, $(L_1, L_2, ..., L_l) = (3, 5, 7, 10, 13...)$. Suppose there is another solution that is different from the current soluction and better than the current one, i.e., use less trucks. Then we can denote the index of the last package of each track by $(L'_1, L'_2, ..., L'_n)$, where $n < l$. For example, $(L'_1, L'_2, ..., L'_n) = (3, 5, 6, 9, 14...)$. Since the new solution is different from the current solution, there must exist an i such that $L'_i < L_i$ and $\forall h \ s.t. \ 0 < h < i, L'_h = L_h$. In our example, $i = 3$. Because the new solution is better than the current solution, there must exist a j such that $i < j \leq n$ and $(L'_{j-1} = L_{j-1}) \wedge (L'_j > L_j)$ or $(L'_{j-1} < L_{j-1} \wedge L'_j > L_j)$. In other words, the new solution pass over the current solution in truck j.

In our example, $j = 5$. However, the first case $(L'_{j-1} = L_{j-1}) \wedge (L'_j > L_j)$ is impossible since it means in the current solution there are more rooms in a truck to accommodate the next package, a contradiction. The second case is also impossible due to the same reason. Therefore there cannot be any other solution that uses less trucks than the current solution. □