Exercises 8

5 March

All questions are unassessed.

1. Let n be a power of 2 ($n = 2^k$). Recall that the recurrence relation for the worst-case number of comparisons for MergeSort, is:

$$\begin{aligned} W(1) &= 0 \\ W(n) &= n - 1 + 2W(n/2) \end{aligned}$$

Prove by induction on k that the solution to this is $W(n) = kn - n + 1$.
(This is the solution which we obtained by repeated expansion - so the induction acts as a check that we got the correct answer).

2. Suppose that Mergesort is to be applied to a list with n elements (where n is a power of two), which is already sorted, but in reverse order.
(a) Write down a recurrence relation for $R(n)$, the number of comparisons required, explaining your answer briefly.
(b) Solve the recurrence relation for $R(n)$.

3. How many comparisons does the Merge algorithm (notes page 62) take to merge two identical sorted lists of length n? Explain your answer.

4. [From the 1999 exam] A sorted list of length $2n$ is disarranged to create a new list L by placing the elements in the odd positions (still in order) before the elements in even positions (still in order). Thus e.g. the list $[1,2,3,4,5,6]$ would become $[1,3,5,2,4,6]$.
How many comparisons does Insertion Sort take when applied to L? Explain your working.

5. For each of the following recurrence relations, draw a recursion tree (see slides 226 and 229) and use the Master Theorem (slides p232) to obtain a solution up to Θ:
(a) $T(n) = 3T(n/4) + 6n$
(b) $T(n) = 5T(n/2) + 2n^2$
(c) $T(n) = 2T(2n/3) + n\log n$
(d) $T(n) = 4T(n/2) + 3n^2$

6. What is the result of applying the split algorithm for Quicksort on slide 236 to the array with keys $[3,5,2,0,8,4,5]$?

7. Explain why the split algorithm for Quicksort on slide 236 takes $n-1$ comparisons on a list of length n. [Hint: what is the variant?]

8. Let E be an array of elements with keys $[6,5,2,7,1,8]$.
(a) Draw the (binary) heap structure corresponding to E.
(b) Use buildMaxHeap to convert E into a max heap (see slide 247). Give your answer both as a tree and as an array.

9. (a) buildMinHeap is defined analogously to buildMaxHeap. Write down the pseudocode for buildMinHeap.
(b) Use buildMinHeap to convert E of Q8 into a min heap. Give your answer both as a tree and as an array.

10. Adapt the algorithms wb2 and wb3 to solve the following problem in both top-down (with memoisation) and bottom-up styles with $O(n^2)$ dictionary lookups. Words in a dictionary have an integer score > 0 given by the function score. The function score returns 0 if the input is a string which is not a word in the dictionary. Given a string s of characters, find the highest score obtainable from splitting the string into words in the dictionary and adding up the scores for the words. If s has no possible splitting into words in the dictionary your algorithm should return a suitable message.

Answers to Exercises 8

1. We will show by induction on k that $W(2^k) = k.2^k - 2^k + 1$ (for $k \geq 0$).
Base case $k = 0$. $W(2^k) = W(1) = 0$ and $k.2^k - 2^k + 1 = 0$. Checked.
Induction step. Assume that for k we have $W(2^k) = k.2^k - 2^k + 1$. We want to show that $W(2^{k+1}) = (k+1).2^{k+1} - 2^{k+1} + 1$. But

$$\begin{array}{rcll} W(2^{k+1}) & = & 2^{k+1} - 1 + 2W(2^k) & \text{by recurrence relation} \\ & = & 2^{k+1} - 1 + 2(k.2^k - 2^k + 1) & \text{by Ind Hyp} \\ & = & (k+1)2^{k+1} - 2^{k+1} + 1 & \text{as required.} \end{array}$$

2. (a)

$$\begin{array}{rcl} R(1) & = & 0 \\ R(n) & = & n/2 + 2R(n/2) \end{array}$$

The list is split into two portions of length $n/2$ which are sorted separately, taking $R(n/2)$ comparisons each. Every element of the LH portion A is greater than every element of the RH portion B. So the merge will compare each element of B with the least element of A, until B is exhausted. This takes $n/2$ comparisons. At the end of this A can be written in without further comparisons.
(b) Let $n = 2^k$:

$$\begin{array}{rcl} R(n) & = & n/2 + 2(n/4 + 2R(n/2^2)) \\ & = & n/2 + n/2 + 2^2 R(n/2^2) \\ \dots & & \\ & = & kn/2 + 2^k R(n/2^k) \\ & = & n\log(n)/2 \end{array}$$

3. $2n - 1$.

4. The list can be represented as $[1,3,..,2n-1,2,4,..,2n]$. The list $[1,3,..,2n-1]$ is already sorted. So each insertion takes one comparison, making $n-1$ comparisons in all. When inserting the element $2i$ ($i = 1,.,n$), the list looks like $[1,2,..,2i-1,2i+1,\dots,2n-1,2i,2i+2,\dots,2n]$. So $2i$ has to be compared with $2n-1, 2n-3,..,2i-1$ ($n+1-i$ comparisons). Hence total is

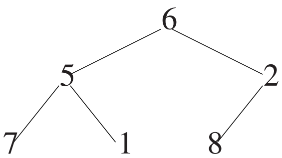
$$n - 1 + \sum_{i=1}^{n}(n+1-i) = n - 1 + \frac{n(n+1)}{2}\ .$$

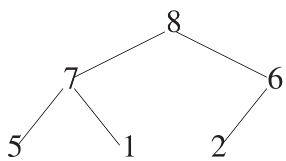
5. The recursion trees are omitted.
(a) Here $a = 3$ and $b = 4$ and $f(n) = \Theta(n)$. So $E = \frac{\log a}{\log b} = \frac{\log 3}{\log 4} = 0.7925$. Set $\varepsilon = 0.2$. Then $n^{E+\varepsilon} = O(f(n))$ and solution is $\Theta(f(n) = \Theta(n)$. The non-recursive work at the root of the recursion tree dominates. Of course any smaller value of $\varepsilon > 0$ would work just as well.
(b) Here $a = 5$ and $b = 2$ and $f(n) = \Theta(n^2)$. So $E = \frac{\log a}{\log b} = \frac{\log 5}{\log 2} = 2.3219$. Set $\varepsilon = 0.3$. So $f(n) = O(n^{E-\varepsilon})$ and solution is $\Theta(n^E) = \Theta(n^{0.7925})$. The recursive work at the leaves of the recursion tree dominates.
(c) Here $a = 2$ and $b = 3/2$ and $f(n) = \Theta(n\log n)$. So $E = \frac{\log a}{\log b} = \frac{\log 2}{\log 1.5} = 1.7095$. Set $\varepsilon = 0.5$. So $f(n) = O(n^{E-\varepsilon})$ and solution is $\Theta(n^E) = \Theta(n^{1.7095})$. The recursive work at the leaves of the recursion tree dominates.

We used the fact that $\log n = O(n^{\varepsilon})$ for any $\varepsilon > 0$.
(c) Here $a = 4$ and $b = 2$ and $f(n) = \Theta(n^2)$. So $E = \frac{\log a}{\log b} = \frac{\log 4}{\log 2} = 2$. So $f(n) = \Theta(n^E)$ and solution is $\Theta(f(n)\log n) = \Theta(n^2 \log n)$. The work done at each level of the recursion tree is roughly equal.

6. Split around first key 3. Resulting array is $[2,0,3,8,4,5,5]$.

7. Variant is $j+1-i$. Initial value right $-$ left $= n-1$. On each iteration of the loop exactly one of (a) i increases by 1 or (2) j decreases by 1. Hence $j+1-i$ decreases by 1. Also the while loop terminates once variant $j+1-i = 0$. Hence the while loop is executed $n-1$ times. But each iteration of the while loop performs exactly one comparison. Hence result.

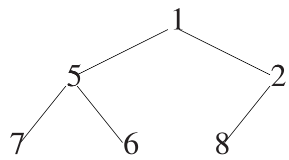
8. (a) 

(b) Max heap $[8,7,6,5,1,2]$. 

9. (a)

Algorithm buildMinHeap(H) scheme:
if H not a leaf:
   buildMinHeap(left subtree of H)
   buildMinHeap(right subtree of H)
   fixMinHeap(H)

Algorithm fixMinHeap(root,heapsize):
left = 2∗root
right = 2∗root+1
if left ≤ heapsize:
   # root is not a leaf
   if left = heapsize:
      # no right subheap
      smallerSubHeap = left
   elif E[left].key < E[right].key:
      # favours right subheap if equal
      smallerSubHeap = left
   else:
      smallerSubHeap = right
   if E[root].key > E[smallerSubHeap].key:
      swap(root,smallerSubHeap)
      fixMinHeap(smallerSubHeap,heapsize)

(b) Min heap $[1,5,2,7,6,8]$. 

10. Memoised recursive solution (top-down):

```
Algorithm Wordscore(s):
procedure ws2(s):
   if len(s) == 0:
      return 0
   else:
      bestscore = −1
      for i = 0 to len(s) − 1:
         wordscore = score(s[i :])
         if wordscore > 0:
            if memo[s[: i]] undefined:
               memo[s[: i]] = ws2(s[: i])
            if memo[s[: i]] ≥ 0 and memo[s[: i]]+wordscore > bestscore:
               bestscore = memo[s[: i]]+wordscore
   return bestscore

memo = {} # empty associative array
ws = ws2(s)
if ws≥ 0:
   return ws
else:
   return ‘no possible splitting’
```

Non-recursive solution (bottom-up):

```
Algorithm ws3(s):
# ws[i] records current best score for s[: i]
# value of −1 indicates no split found (yet)
n = len(s)
ws[0] = 0
if n > 0:
   for i = 1 to n:
      ws[i] = −1
      for j = 0 to i − 1:
         wordscore = score(s[j : i])
         if ws[j] ≥ 0 and wordscore > 0:
            if ws[j] + wordscore > ws[i]:
               ws[i] = ws[j] + wordscore
if ws[n] ≥ 0:
```

```
   return ws[n]
else:
   return 'no possible splitting'
```