Exercises 6

19 February

All questions are unassessed.

1. (adapted from Baase) Here is the Binary Search algorithm as presented in the lecture notes:

Algorithm Binary Search (BS):
procedure BinSearch(left, right):
  # searches for x in L in the range L[left] to L[right]
  if left $>$ right:
    return "not found"
  else:
    mid $:= \lfloor (\text{left} + \text{right})/2 \rfloor$
  cond
    $x = L$[mid]: return mid
    $x < L$[mid]: BinSearch(left, mid $-$ 1)
    $x > L$[mid]: BinSearch(mid $+$ 1, right)

Give a modified version of the Binary Search algorithm which eliminates unnecessary work given that it is known that x is definitely in the list. Draw a decision tree for the modified algorithm for $n = 7$. What is the worst-case number of comparisons for $n = 7$?

2. (a) Draw a decision tree for (the usual) binary search on an ordered list of length 10. How many comparisons does the algorithm take in worst case? Calculate the average number of comparisons, on the basis that the element being searched for is in the list, and all positions are equally likely.
(b) Now assume that the element being searched for is known to be in the list (still of length 10), and that all positions are equally likely. Calculate the average number of comparisons for the modified version of binary search where we take advantage of knowing that the element is in the list (see Q1).

3. (a) Give a decision tree for an algorithm which, given three distinct integers, finds (using pairwise comparisons) the middle one under the usual ordering. Your algorithm should be optimal in the sense that it uses the fewest comparisons in the worst case. How many comparisons does your algorithm use in worst case?
(b) Explain why an algorithm which uses fewer comparisons in worst case could not compute the middle integer correctly in every case.

4. (Baase) Write an algorithm to find the second-largest element in a list containing n entries. How many comparisons of list entries does your algorithm do in the worst case? Experiment to see if you can improve on the performance of your algorithm, trying particular values of n such as $n = 3$, $n = 4$.

5. You are given five coins of identical appearance. They all weigh the same, apart from one coin, which is counterfeit and weighs a different amount from the others. Your task is to identify the counterfeit coin. You are given scales of the balance type, allowing you to compare the weights of the coins against each other. Each weighing has two outcomes, i.e. the weights are equal or unequal.

(a) Give an algorithm to solve the problem in the form of a decision tree, with the internal nodes representing possible weighings, and the leaves representing the possible results. Your algorithm should be optimal, in the sense that it uses no more weighings in worst case than necessary.
(b) State how many weighings your algorithm uses in worst case.
(c) Explain why your algorithm is optimal, i.e. explain why there is no algorithm which uses fewer weighings in worst case than your own. [Hint: consider the leaves of the decision tree]
(d) Assuming that each of the five coins is equally likely to be the counterfeit one, calculate the average number of weighings which your algorithm uses.

6. Let

$$\begin{aligned} f(n) &= 6n^2 - n + 5 \\ g(n) &= n^2/4 + 13n - 2 \end{aligned}$$

Show that $g \in O(f)$, $f \in O(g)$.

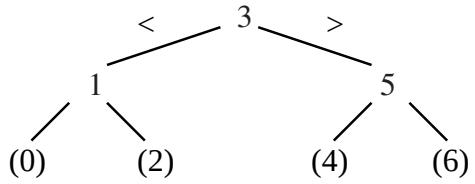
7. Solve the following recurrence relation (see notes page 53):

$$\begin{aligned} F(1) &= 5 \\ F(n) &= 3 + F(\lfloor n/2 \rfloor) \end{aligned}$$

Answers to Exercises 6

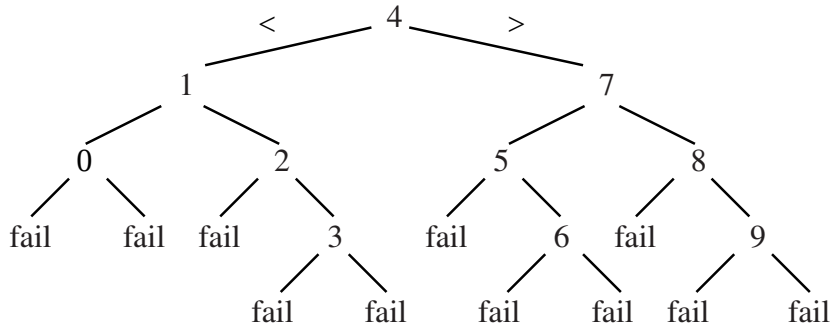
1.

Algorithm Modified Binary Search:
procedure ModBinSearch(left, right):
  # searches for x in L in the range L[left] to L[right]
  if left = right: # will never have left > right
    return left
    # This is the modification: if left = right then know that have found answer
  else:
    mid := $\lfloor$(left + right)/2$\rfloor$
  cond
    $x = L$[mid]: return mid
    $x < L$[mid]: ModBinSearch(left, mid − 1) # will not arise if mid = left
    $x > L$[mid]: ModBinSearch(mid + 1, right)

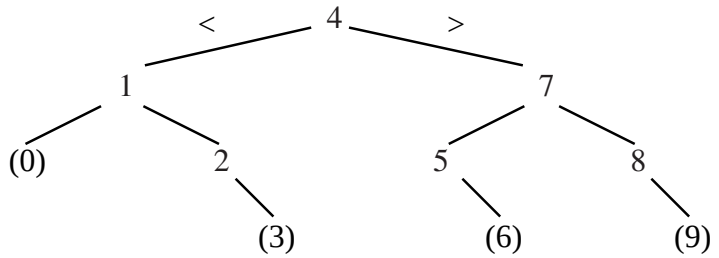


()s indicate that a comparison does not need to be made. Worst-case number of comparisons 2.

2. (a) We index the list from 0 to 9. Also we always choose the left-hand index where there is a choice (i.e. we take the *floor* of (left + right)/2 rather than the *ceiling*). 4 comparisons in worst case.
4 takes 1 comp. 1,7 take 2. 0,2,5,8 take 3. 3,6,9 take 4. Average no of comparisons = $(1\times1+2\times2+4\times3+3\times4)/10 = 2.9$.

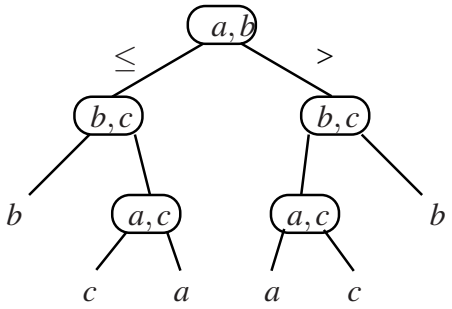


(b) If 4 takes 1 comp. If 0,1,7 takes 2 comps. If 2,3,5,6,8,9 takes 3 comps. Average no of comparisons = $(1\times1+3\times2+6\times3)/10 = 2.5$.



The numbers in parentheses indicate that a result is returned without a further comparison.

3. (a) For instance (cf notes page 43)

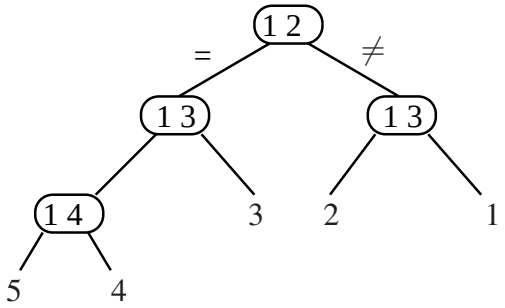


3 comparisons in worst case.
(b) Method 1. Any better algorithm A would have to use no more than 2 comparisons. We can suppose without loss of generality that A starts by comparing a with b. We can also suppose that $a < b$ (by adjusting the input if necessary). For the second comparison A must either compare a, c or b, c (since a repeat of a, b achieves nothing). Moreover A must choose purely on the basis of knowing $a < b$. But if A compares a, c it may find that $a < c$, in which case A cannot determine whether b or c is the middle integer. And if A compares b, c it may find that $c < b$, in which case A cannot determine whether a or c is the middle integer.
Method 2. We show that finding the middle element involves sorting the list. This is because we cannot discover that y is the middle element without comparing it with one of the other two. But then the knowledge that y is second together with this comparison is enough to sort the list. We know on general grounds that 3 comparisons are needed in worst-case to sort a 3 element list.
Method 3. We claim that after a single comparison, all three outcomes are still possible (compare the diagram in part (a)). We see this as follows: Suppose without loss of generality we start by comparing a with b. If $a < b$ then the order of the three elements could be $[c, a, b]$ or $[a, b, c]$ or $[a, c, b]$. So any one of a, b, c can be the middle element. This is equally true if $b < a$.
Now for a binary tree to have at least 3 leaves (for the three outcomes), it must have depth ≥ 2. So after the first comparison both the left and right subtrees have depth ≥ 2. Hence the whole decision tree has depth ≥ 3, which shows that 3 comparisons are needed in the worst case.

4. An obvious algorithm is to find the largest element using e.g. linear search (Findmax), and then find the maximum of the remaining elements, by the same method. For a list of size n this takes $(n-1)+(n-1-1) = 2n-3$ comparisons in every case.
For $n = 3$ this method takes 3 comparisons (which is in fact optimal). For $n = 4$ our algorithm takes 5 comparisons. Can we do better? In fact there is an better algorithm which takes only 4 comparisons. This is as follows:

Play a knockout tournament between the elements. We notice that the second best player must have been beaten by the best (though possibly quite early on in the tournament). Hence we simply have to find the best of the $\log n$ players that the best one has beaten. This can be done in $(\log n) - 1$ so that the total number of comparisons is $n + \log n - 2$. For $n = 8$ we have improved from 13 to 9. Of course both algorithms are $\Theta(n)$. (For simplicity we here assume that n is a power of 2.)

5. (a) There are many possible decision trees.



(b) 3.
(c) The tree must have 5 leaves (for the 5 different results). A tree of depth ≤ 2 can have at most 4 leaves. Hence the tree must have depth at least 3, giving a lower bound of 3 comparisons in worst case.
(d) The answer will depend on the tree given in answer to (a). For the one given above, average number of weighings is the average distance to the leaves, i.e.

$$(3+3+2+2+2)/5 = 12/5$$

[General theory indicates that this is the smallest possible value (the tree is balanced).]

6. Clearly $13n \leq n^2$ for $n \geq 13$. Hence $g(n) = n^2/4 + n^2 \leq 2n^2$ for $n \geq 13$. But since $n \leq n^2$ (all n) we have $f(n) \geq 5n^2$. Hence $g(n) \leq 2n^2 \leq f(n)$ for $n \geq 13$. So $g \in O(f)$.
Now $5 - n \leq n^2$ for $n \geq 2$. Hence $f(n) \leq 7n^2$ for $n \geq 2$. But $g(n) \geq n^2/4$ for $n \geq 1$. Combining we have $f(n) \leq 7n^2 \leq 7.4.g(n)$ for $n \geq 2$. Hence $f \in O(g)$.
In fact both f, g are plainly $\Theta(n^2)$, and the above working, where n^2 is sandwiched between f and g, shows this.

7.

$$\begin{aligned} F(n) &= 3 + F(\lfloor n/2 \rfloor) \\ &= 3 + 3 + F(\lfloor n/4 \rfloor) \\ &\cdots \\ &= 3 + 3 + \cdots + 3 + F(1) \\ &= 3\lfloor \log(n) \rfloor + 5 \end{aligned}$$