

Degree Examination

MX4028 Algorithms

Friday 28 May 1999

(9am to 11am)

Attempt *THREE* questions

Calculators may be used *ONLY* for the arithmetic of real numbers or the numerical evaluation of trigonometric, logarithmic and exponential functions. Calculator memories must be clear at the start of the examination; in particular, the use of pre-stored programs is prohibited. Marks may be deducted for answers that do not show clearly how the solution is reached.

1. (a) Describe the “Quicksort” algorithm for sorting a list and give pseudo-code for a routine

quicksort(x, lo, hi)

which sorts the sublist x_{lo}, \dots, x_{hi} of the list \mathbf{x} . You may assume that the routine `separate(x, lo, hi, sep)`, in which a list is separated into two sublists based on the separator `sep`, is available. Illustrate your answer by “quicksorting” the list

6 4 8 2 11 5 3 7 9 1 10

[When making a “random” choice from a list, choose the item that occurs in the first position, so a “random” choice from $[2, 1, 3]$ would be 2.]

(b) Let T_n be the average time taken to quicksort a list of n elements. Assuming that the separation algorithm applied to a list of length n takes time αn , derive the recurrence relation

$$T_n = \alpha n + \frac{2}{n} \sum_{k=0}^{n-1} T_k,$$

explaining what other simplifying assumptions have been made. Hence show that

$$\frac{1}{n+1} T_n = T_0 + \sum_{k=0}^{n-1} \frac{\alpha(2k+1)}{(k+1)(k+2)}.$$

(c) How does Quicksort compare with insertion sort?

2. Give a brief description of a priority queue and a complete binary tree. What does it mean to say that a complete binary tree satisfies the *heap condition*?

Describe the use of Heapsort to sort the string

HEAPSORT

Give the full construction of the heap in detail, and give details of the first step in its use.

Does Heapsort have any advantages over Quicksort?

3. Consider the formal language \mathcal{L} given by the following grammar.

- The alphabet is $A = \{a\}$.
- The abstract alphabet is $\mathcal{A} = \{\sigma, \alpha, \beta, \gamma\}$.
- The initial symbol is σ .
- The productions are
 1. $\sigma \rightarrow \alpha\beta$;
 2. $\alpha\beta \rightarrow \alpha\alpha\beta$;
 3. $\alpha\beta \rightarrow \alpha\alpha\gamma$;
 4. $\alpha a \rightarrow a\alpha a$;
 5. $\alpha\alpha\gamma \rightarrow \alpha\gamma$; and
 6. $a\alpha\gamma \rightarrow a$.

(a) Prove that the string a^2 is in \mathcal{L} .

(b) Prove that the string a^4 is in \mathcal{L} .

(c) Let s be a string generated from σ . For each a in s , define the *weight* of a to be 2^k , where k is the number of copies of α which occur in s to the left of a . Show that the sum of the weights of all copies of a in s is the same before and after an application of production (4).

(d) Give a simple description of the strings in \mathcal{L} and give arguments to support your claim. [A formal proof is not expected.]

4. In this question, you are given a random number generator `RAND()` which produces random reals uniformly distributed in $[0, 1)$.

(a) You are required to choose k objects “at random” from a sequence of N objects — perhaps passing on a conveyor belt. You must decide whether a given object is to be selected *before* the next one becomes available. Give pseudocode for such an algorithm, and justify the assertion that it will produce exactly k items. [You are *not* asked to justify that the sample is a random sample.]

(b) Let T be the triangle with vertices at $(0, 2)$, $(-1, 0)$ and $(1, 0)$. Describe an algorithm to generate a sequence of points $\{p_n\}$ in T “at random”. Your answer should produce an additional point p_n for each pair of calls to `RAND()`. Explain briefly why your algorithm does what is required.

Degree Examination

SOLUTIONS

MX4028 Algorithms

Friday 28 May 1999

(9am to 11am)

1. (a) We start with our list $\{x_1, \dots, x_n\}$, pick some element in the list, called the *separator*, and then rearrange the list so that all the elements that are less than or equal to the separator come before it and all the elements that are greater than the separator come after it. Having done this we then recursively apply the algorithm to each of these sublists. The recursion continues along any branch until its sublist shrinks down to zero or one element. The outline of the program is:

```

algorithm quicksort(x,lo,hi)
// to sort  $x_{lo}, \dots, x_{hi}$ 
begin
  if hi > lo begin          // i.e. if there is anything to sort
    choose separator s from list
    separate(x,lo,hi,sep) // separate out the list into the form
                          //  $(x_{lo}, \dots, x_k, \quad s \quad x_{k+2}, \dots, x_{hi})$ 

    quicksort(x,lo,k)
    quicksort(x,k+2,hi)
  end
end

```

The whole list is then sorted with a call to `quicksort(x,1,n)`. The choice of s can be made at random from the list.

We show the various stages in sorting the given list, drawing a box round the separator element. At each stage the list is shown after the call to `separate` and before recursive calls to the two sublists.

6	4	8	2	11	5	3	7	9	1	10
4	2	5	3	1	6	8	11	7	9	10
2	3	1	4	5	6	7	8	11	9	10
1	2	3	4	5	6	7	8	9	10	11
1	2	3	4	5	6	7	8	9	10	11

Note that the last sort was necessary; although the previous line was in fact sorted, this depended on the accidental positioning of 10 after 9 in the original array.

- (b) We assume that the initial list is a permutation of $\{1, \dots, n\}$ and that all permutations are equally likely. Suppose the list splits into sublists of lengths k and $n - 1 - k$; this leaves one left over for the separator. Then on average

$$T_n = \alpha n + T_k + T_{n-1-k}$$

By our assumptions, all possible values of k are equally likely. We thus remove the dependence on k by taking the average over all the possibilities:

$$T_n = \alpha n + \frac{1}{n} \sum_{k=0}^{n-1} (T_k + T_{n-1-k}).$$

Since each of the T_k 's appears twice in the sum, we obtain

$$T_n = \alpha n + \frac{2}{n} \sum_{k=0}^{n-1} T_k$$

as required. Using this relationship twice,

$$\begin{aligned} \frac{n}{2}(T_n - \alpha n) &= T_0 + \cdots + T_{n-1} \\ \frac{n+1}{2}(T_{n+1} - \alpha(n+1)) &= T_0 + \cdots + T_{n-1} + T_n. \end{aligned}$$

Subtracting these and rearranging gives

$$\frac{T_{n+1}}{n+2} = \frac{T_n}{n+1} + \frac{\alpha(2n+1)}{(n+1)(n+2)}.$$

(c) Although insertion sort is simple to program without error, it runs in time $O(n^2)$, where n is the number of elements in the list. Although more complicated as we saw above, Quicksort has an average running time of $O(n \log n)$, and is very significantly faster for long lists than insertion sort.

2. A priority queue is a collection of elements or items, each of which has an associated priority. The operations available are:-

`create` creates an empty priority queue;

`add(item)` adds the given `item` to the priority queue; and

`remove` removes the item with the highest priority from the queue.

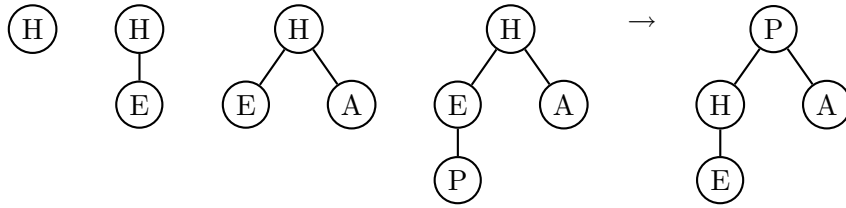
A complete binary tree is a tree which is either empty, or one in which every node:

- has no children; or
- has just a left child; or
- has both a left and a right child.

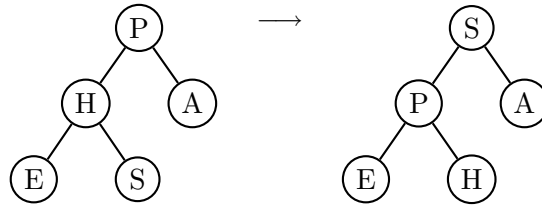
In addition, we require that all the levels, except perhaps the last, has both a left and a right child; while on the last level, any missing nodes are to the right of all the nodes that are present.

If we have a representation of a priority queue as a complete binary tree in which each node contains an element and its associated key or priority, it satisfies the *heap condition* if at each node, the associated key is larger than the keys associated with either child of that node.

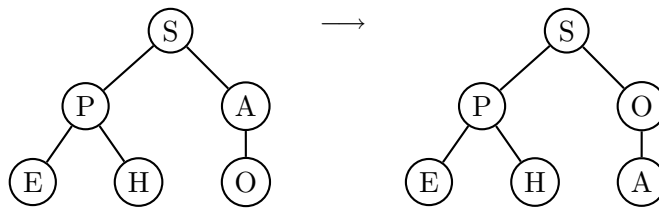
We now create a priority queue, represented as a complete binary tree, in which letters at the end of the alphabet have priority. When the first three characters are added as additional nodes in the binary tree, the new tree already satisfies the heap condition. Adding “P” violates that condition; “P” thus has to rise until it is again satisfied.



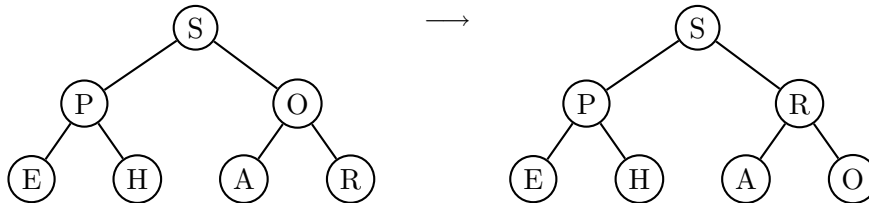
When “S” is added, it rises to the top of the tree before the heap condition holds.



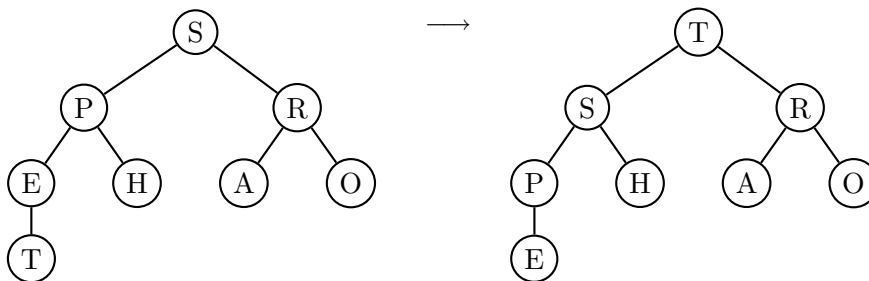
Now add “O”; it has to rise to be above “A”



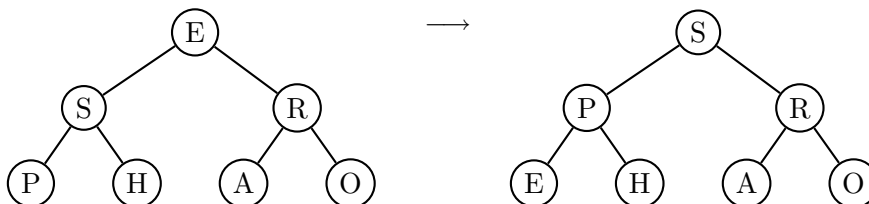
Now add “R”; this replaces “O” and fills the tree at level 2.



The final addition of “T” completes the tree, when it has been re-heaped.



The string is now sorted by removing the root node — this is the item with the highest priority remaining in the tree — and replacing it with the “last” node in the tree. The tree no longer satisfies the heap condition, and so the node that has just been promoted is allowed to fall again until the condition holds.



The root of the tree again contains the next character in order and is removed. This continues until the whole heap has been destroyed.

Finally, note that although in general Heapsort is slower than Quicksort by a factor of about 2, the “worst case” behaviour remains $O(n \log n)$, whereas Quicksort can become $O(n^2)$.

3. (a)

$$\sigma \xrightarrow{(1)} \alpha\beta \xrightarrow{(3)} \alpha a \gamma \xrightarrow{(4)} a^2 \alpha \gamma \xrightarrow{(6)} a a$$

and so a^2 is in \mathcal{L} .

(b)

$$\sigma \xrightarrow{(1)} \alpha\beta \xrightarrow{(2)} \alpha\alpha\beta \xrightarrow{(3)} \alpha^2 a \gamma \xrightarrow{(4)} \alpha a^2 \alpha \gamma \xrightarrow{(4)} a^2 \alpha a \alpha \gamma \xrightarrow{(4)} a^4 \alpha^2 \gamma \xrightarrow{(5)} a^4 \alpha \gamma \xrightarrow{(6)} a^4$$

and so a^4 is in \mathcal{L} .

(c) Consider the effect of applying a modified form of production (4) in which αa is replaced by $a \alpha$. The weight associated with this copy of a reduces from 2^{k+1} (say) to 2^k and the weights of all other a 's remains the same. Now consider the effect of using the correct form of production (4). In addition to that loss of weight, an additional a is introduced, which also has weight 2^k . This restores the total weight of the string to what it was originally.

(d) Note that production (1) is a forced first step and that afterwards there is exactly one β until production (3) is made, at which point there are no copies of β and only the last three productions are relevant. At this stage, the string has the form $\alpha^n \beta$ for some $n \geq 1$. We can get all such with $n \geq 1$ simply by applying production (2) a total of $n - 1$ times.

Counting γ 's shows we only use production (6) once, as the last step to make the string fully concrete. Until the string becomes fully concrete, γ remains as the last element. Thus production (5) can only be used at the right-hand end of the string, and so does not effect the total weight. At the start of this phase the total weight is 2^n . Just before we apply production (6), the string must be in the form $a^k \alpha \gamma$, and this has total weight k . Thus the final production is

$$a^{2^n} \alpha \gamma \xrightarrow{(6)} a^{2^n}$$

and $\mathcal{L} = \{a^{2^n} \mid n \geq 1\}$. This proof was not particularly easy; an indication of some similar argument would be acceptable.

4. (a) We give the pseudocode for the Running Sample algorithm in Fig. 1

The logic of this method is that if, at the i th item, we have so far chosen j items then we have $k - j$ items left to choose from the remaining $N - i + 1$ items (including the i th). So it seems reasonable to choose the i th item with probability

$$\frac{k - j}{n - i + 1}$$

and that's what the algorithm does.

We cannot end up with more than k items because we stop the loop if we have got k . It is thus enough to show that we cannot end up with *fewer* than k . Suppose conversely that we had ended up with $j < k$ items and that the last item *not* chosen was the i th. Then, at that stage, we had already chosen $j - (N - i)$ items. So the probability of choosing the i th was

$$\frac{k - (j - (N - i))}{N - i + 1} = \frac{N - i + (k - j)}{N - i + 1} \geq 1,$$

```

algorithm runningsample(k, N)
// choose k items at random from N supplied
begin
  t = 0
  chosen = 0
  repeat begin
    if ((N-t)*rand() >= (k-chosen)) begin
      // pass over next item
      read in next item
      t = t + 1
    end
    else begin
      // accept next item
      read in next item
      print out next item
      t = t + 1
      chosen = chosen + 1
    end
  end // repeat
until (chosen = k)
end.

```

Figure 1: Pseudocode for the Running Sample algorithm.

so we *must* have chosen it! Contradiction.

(b) Our algorithm identifies the half of the equilateral triangle to the left of the x - axis with the remaining half of the square

$$S = \{(x, y) \mid 0 \leq x \leq 1, 0 \leq y \leq 2\},$$

namely the half above the line $y + 2x = 2$. Our algorithm is then

```

x = RAND();
y = RAND();
if ( y > 2 - 2*x) then
  x = -1 + x
  y = 2 - y
endif
return (x,y)

```

The pair (x, y) are chosen uniformly from a rectangle with the same area as the given triangle. Our algorithm arranges to use all the points produced, while the transformation which maps the part of the square outside the equilateral triangle to the “missing” part of the triangle is Euclidean and hence preserves areas.

Degree Examination

SETTER'S COMMENTS

MX4028 Algorithms

Friday 28 May 1999

(9am to 11am)

This is the third time the option has run in recent years, and the second time as a reading course. I have tried to focus either on the main topics, or on things they should find easy using the skills they practiced during the course.

1. This is straightforward and a repeat of June 1997. It is all book-work, and was all emphasised. I wanted to have some work on counting included, and hope this is relatively easy. I expect them to use “my” random number generator in the example, since the last sort used in it raises an important point.
 - (a) The algorithm [6 marks]
 - (b) The example; there is scope to pick up marks missed in the first part if the point is made in the example. [6 marks]
 - (c) Timing; deriving the formula for T_n . [6 marks]
 - (d) Comparison with insertion sort. [2 marks]

2. This is the first time I've set a “Heapsort” question. I've tried to make it straightforward. I would hope that, if they ever understood the algorithm, they could work it out from the hints. Of course I will take anything like this, am not worried about which order is used etc. This is like an example and exercise in their notes.
 - the first bookwork, [6 marks]
 - heap creation, [6 marks]
 - heap destruction, why the sort works, and comparison. [8 marks]

3. Hmmm! There has been a “language” question for the last two years, but it appeals to my taste as being a programming language that mathematicians don't recognise as such!

I know this one is hard, and I will try to be as generous as possible.

 - (a) Getting a^2 is a warm-up exercise while they understand the productions. [4 marks]
 - (b) And a^4 uses all the power, but is close to “follow your nose?” [6 marks]
 - (c) The induction step is really here to provide the right tool to do the last bit. But I expect them not to understand. [4 marks]
 - (d) Without the concept of weight, this problem can be *hard* to get all the cases. I'll give significant credit for showing we can get each a^{2^n} , although it is the converse that is hard. [6 marks]

4. This is the first time I've set a random number question, although it is one of the six chapters in the notes.

(a) Conveyor belt selection is all bookwork; I hope it is obvious too! [10 marks]

(b) The equilateral triangle is new, although they have done similar, and harder questions. It shouldn't be too taxing, but there is scope for error here! [10 marks]