COS 126 General Computer Science Spring 1999

Midterm 1 - Solutions[1]

Question 1

```
i = 0;
while (i < N)
  {
    a += i;
    i += 2;
  }
```

The body of the `for` loop gets executed *before* the increment step.

Question 2

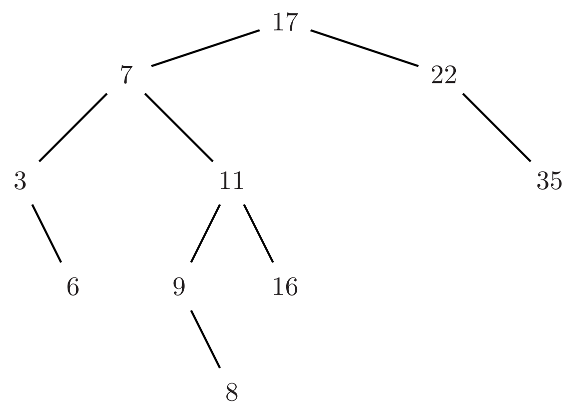
```
link x;

x = list->next;            // store pointer to second node
list->next = x->next;      // make first node point to third node
x->next = list;            // make second node point to first node
list = x;                  // update pointer to first node of list
```

The order that these statements get executed is very important.

Question 3

(a) 5, (b) 2, (c) 4



- To search for the key 8, we start from the root. We compare 8 with 17. Since $8 < 17$, we know that 8 would have to be in the left subtree (the tree rooted at 7). Now, since $8 > 7$, we know that 8 would have to be in the right subtree of node 7, i.e., the subtree rooted at 11. Since $8 < 11$, we consider the subtree rooted at 9. Since $8 < 9$, we consider the subtree rooted at 8. Finally we discover 8 == 8. This required 5 comparisons.

[1]Copyright 1999, COS 126.

- Using the same logic, we see $19 > 17$, so we consider the subtree rooted at 22. Then since $19 < 22$, we would then explore the left child of 22. But it is NULL, so we conclude that 19 is not in the BST.
- Similarly, $10 > 7$, so we consider the subtree rooted at 7. Since $10 < 11$, we consider the subtree rooted at 9. Since $10 > 9$, we would then explore the right child of 9. But it is NULL, so we conclude 10 is not in the BST.

Question 4

(a) L T A, (b) A T L or T A L, (c) A L T

- With arrays, we store the N keys sequentially in memory. Hence, it is fast to access the kth element. If the keys are sorted, we can search for a given element in at most $\log_2 N$ steps using binary search. The drawback of arrays is that inserting and deleting keys is slow. To delete the smallest key, we would have to shift over the other N-1 keys.
- With linked lists, we store the N keys in arbitrary memory locations, and maintain a pointer `list` to the beginning of the list. Each node stores a pointer to the next node; these N pointers take up extra space. Deleting the first node can be done with a single step - we simply change `list` to point to `list->next`. Searching can be time consuming with a linked lists, since it is not easy to access the kth element. Instead, we would have to traverse the list until we found the key (or a larger key).
- Binary search trees provide the best of both worlds - fast search and fast insert/delete. Now, for each node we maintain two different pointers to other tree nodes; these extra $2N$ pointers require additional storage. We also maintain a pointer `root` to the root of the tree. The *binary search tree property* is maintained: for each node `x`, all keys in its left subtree are less than `x->key` and all keys in its right subtree are greater than `x->key`. This allows for a fast sorting procedure. The number of steps to search for an existing key is equal to that key's depth in the tree. If the tree is reasonably balanced, this will be $O(\log N)$. Also, it is possible to insert and delete keys in $O(\log N)$. Deleting the smallest key is particularly easy, since it must be at the very bottom of the tree, and can just be removed without ruining the binary search tree property. To find the smallest key, go down the left-side of the tree. This will take $O(\log N)$ steps if the tree is balanced.

Question 5

Here's the truth table:

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Don't forget that 0 is even since 0 is a multiple of 2. The sum-of-products method gives: $x'y'z' + x'yz + xy'z + xyz'$. Use your imagination to view the picture.

Question 6

1 2 4 8
$O(N)$ time

The trickiest part is realizing that the function is called recursively *before* the variable `value` is printed. Thus, the first value to get printed will be 1. To evaluate `f(N)`, the function evaluates `f(N-1)` Then to evaluate `f(N-1)`, the function evaluates `f(N-2)`. Eventually, the recursion will "bottom-out" when `N == 0`. Thus, there will be `N` recursive calls.

Question 7

```
4 2 8 6 5 3 7 1
---------------
2 4 8 6 5 3 7 1
2 4 8 6 5 3 7 1
2 4 6 8 5 3 7 1
2 4 5 6 8 3 7 1
2 3 4 5 6 8 7 1
2 3 4 5 6 7 8 1
1 2 3 4 5 6 7 8
```

The easiest and quickest way to answer this question is to realize that the code provided is insertion sort. Note that the `printf` statement prints the array contents each time through the i loop.

The outer loop i ranges from 1 to $N-1$. The inner loop ranges from 1 to i. Thus the total number of comparisons is exactly $1+2+\ldots+N-1 = N(N-1)/2$ or $O(N^2)$.

Question 8

```
    int i, count[10];

    for (i = 0; i < 10; i++)        // initialize
      count[i] = 0;

    for (i = 0; i < N; i++)
      count[scores[i]/10]++;        // integer arithmetic

    for (i = 0; i < 10; i++)
     printf("[%2d-%2d]: %d\n", 10*i, 10*i+9, count[i]);
```

This is best done using an array `count[10]`, where the ith element in the number of midterm scores in the ith range (exactly like on the Mandelbrot assignment). Using integer arithmetic is the easiest way to translate from midterm score to interval number.

You could solve this problem using a sequence of `if-then-else` statements, but this would consume valuable time and is unnecessarily complicated. If you find yourself writing essentially the same lines of code over and over, use a loop; if you are writing essentially the same variable names over and over, use an array!

```
     int bin0, bin1, bin2, bin3, bin4, bin5, bin6, bin7, bin8, bin9;
     int i;

     bin0 = 0; bin1 = 0; bin2 = 0; bin3 = 0; bin4 = 0;
     bin5 = 0; bin6 = 0; bin7 = 0; bin8 = 0; bin9 = 0;

     for (i = 0; i < N; i++)
       {
         if (scores[i] < 10) bin0++;
         else if (scores[i] < 20) bin1++;
         else if (scores[i] < 30) bin2++;
         else if (scores[i] < 40) bin3++;
         else if (scores[i] < 50) bin4++;
         else if (scores[i] < 60) bin5++;
         else if (scores[i] < 70) bin6++;
         else if (scores[i] < 80) bin7++;
         else if (scores[i] < 90) bin8++;
         else bin9++;
       }

     printf("[00-09]: %d\n", bin0);
     printf("[10-19]: %d\n", bin1);
     printf("[20-29]: %d\n", bin2);
     printf("[30-39]: %d\n", bin3);
     printf("[40-49]: %d\n", bin4);
     printf("[50-59]: %d\n", bin5);
     printf("[60-69]: %d\n", bin6);
     printf("[70-79]: %d\n", bin7);
     printf("[80-89]: %d\n", bin8);
    printf("[90-99]: %d\n", bin9);
```