# Computer Science Foundation Exam

**August 12, 2005**

# Computer Science

# Section 1B

## No Calculators!

**KEY**

**Solutions and Grading Criteria**

**Name:** _____

**SSN:** _____

**Score:** 50

**In this section of the exam, there are four (4) problems. You must do all of them.**

**The weight of each problem in this section is indicated with the problem.**

**Partial credit cannot be given unless all work is shown and is readable.**

**Be complete, yet concise, and above all be neat.**

**1. [12 points]**

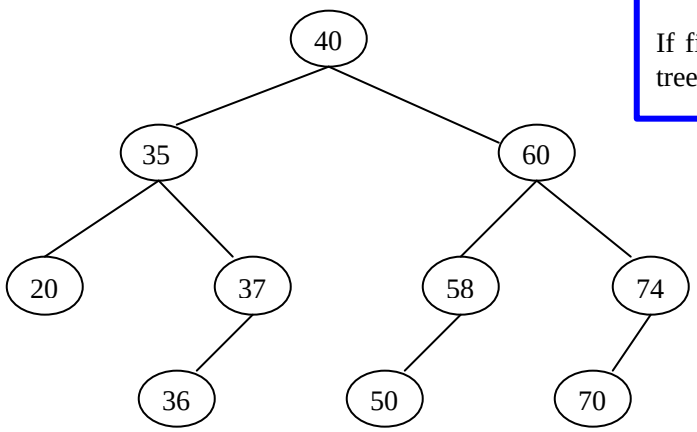
**(a – 6 points)**

Construct a binary search tree from the values shown below. The values appear in the order in which they are input to the binary search tree construction process.

Input Values: 40, 35, 60, 74, 58, 37, 70, 36, 20, 50

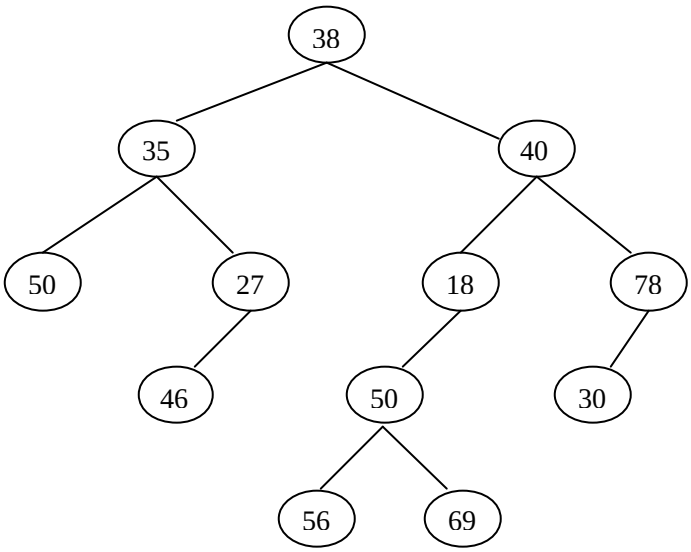
Tree must be drawn in the order the values appear. If another order is used deduct entire 6 pts.

If final tree is not a valid search tree deduct entire 6 pts.




**(b – 6 points)**

List the order in which the nodes in the tree shown below are visited in a postorder traversal.




Deduct 2 pts for first value out of order and 1 pt for each additional value out of order.

The order the nodes are visited in postorder traversal is:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 46 | 27 | 35 | 56 | 69 | 50 | 18 | 30 | 78 | 40 | 38 |

**2. [16 points]**

The selection sort algorithm sorts an array of n elements as follows (subscripts from 0 to n-1): locate the largest element in the array, then switch the largest element with the element at subscript n-1, thereby placing the largest element in the last position. Then locate the largest element remaining in the subarray with subscripts 0 to n-2 and switch it with the element at subscript n-2, thereby placing the second largest element in the next to last position. The process repeats until the array is sorted. Write a **recursive** selection sort function that will correctly sort an array of n elements. Assume that the elements are integer values. Hint: a helper function might be useful.

**One possible solution is:**

```
void locate_max (int array[], int n)
{
     int temp,    /* temp use for exchange of position */
         max_index;  /* index of largest element */

     max_index = n-1;  /* assume last value is largest */
     for (int j = n-1; j>= 0; --j)
          if (array[j] > array[max_index])
               max_index = j;

     /* do exchange if necessary */
     if (max_index != n-1)
     {   temp = array[n-1];
         array[n-1] = array[max_index];
         array[max_index] = temp;
     }
}


void select_sort (int array[], int n)
{
     if (n > 1)
     {   locate_max(array, n);
         select_sort(array, n-1);
     }
}
```

There is certainly more than one way to solve this problem, the solution given is simply one technique. Make sure their code is recursive and deduct the entire 16 pts if it is not recursive.

Be sure to check the stopping conditions.

**3. [12 points]**

Show the exact output produced by the following program by tracing its execution. Place you answer in the box provided below.

```
#include <stdio.h>
int q1 (int *, int, int *);
int q2 (int *, int *, int);
int main ( ) {
    int a= 4;
     int b = 17;
     int *c;
     c = &b;
     a = q1(&a, b, c);
     printf("2. a = %d, b = %d, *c = %d\n",a, b, *c);
     c = &a;
     a = q2(c, &b, a);
     printf("4. a = %d, b = %d, c = %d\n", a, b, *c);
     return 0;
}
int q1 (int *x, int y, int *z){
     int a = 5;
     int *p;
     printf("1. *x = %d, y = %d, *z = %d\n",*x, y, *z);
     p = z;
     *p = a + *x;
     return (*x + *z + y);
}
int q2(int *x, int *y, int z) {
     int b = 3;
     int *p;
     int *q;
     q = y;
     p = x;
     *p = *x - *q - b;
     b = *p;
     p = q;
     *q = 6;
     printf("3. *x = %d, *y = %d, z = %d\n", *x, *y, z);
     p = x;
     *p = b - *y;
     return(*x + *y - 4);
     }
```

Exact output is:

1. *x = 4, y = 17, *z = 17
2. a = 30, b = 9, *c = 9
3. *x = 18, *y = 6, z = 30
4. a = 14, b = 6, *c = 14

If they are not careful, it is easy to get off track when tracing, especially if they are sloppy in their technique. Since we are looking for exact output here, precision is important. Count each output as 1 point.

**4. [10 points – 5 points each part]**

Given the following program answer the questions (a) and (b) below.

```
#define n 6

void arrayOp(int X[]);

int main(void)
{
  int X[n];
  X[0] = 1; X[1] = 3; X[2] = 1; X[3] = 2; X[4] = 1; X[5] = 2;
  arrayOp(X);
  return 0;
}

void arrayOp(int X[])
{   int i, j, k, s, t;
    i = 0;
    while (i < n)
    {   i++;
        s = 0;
        j = 1;
        while (j <= n)
        {   s = X[j] + s;
            j++;
        }
        X[i] = s;
   }
}
```

Simple tracing exercise using arrays. Again they must be careful and probably the ones who are not will miss the last statement in the outer loop which sets the value for the next array element since index "i" is incremented just inside this loop.
Part (a) – 1 point per answer.
Part (b) - make sure the answer is expressed in terms of "m", if they did the trace correctly, this part should be correct.

(a) Show the values in array X after the function has completed execution.

**Array X after function execution**

| index | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| value | 1 | 9 | 15 | 29 | 56 | 111 |

(b) Consider the case where each element of the array X is equal to some value `m`, where `m` is a positive integer. What will be the sum of the elements of array X, in terms of `m`, when the program finishes?

`1m + 5m + 9m + 17m + 33m + 65m = 130m`