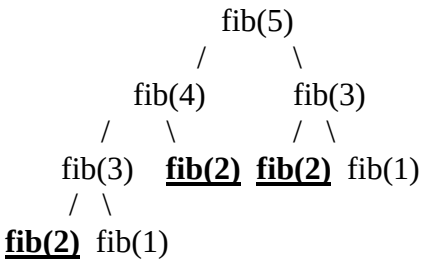
**Computer Science I Honors**
**Exam 1**
**Practice**

1) When executing the call fib(5), where fib is the recursively defined function that returns the corresponding Fibonacci number, how many recursive calls are made to fib(2)?

**3 calls**




2) Determine the closed form solution of the following summation:

$\sum_{i=5}^{2n}(3i-10)$

$= \sum_{i=5}^{2n}(3i) - \sum_{i=5}^{2n} 10 \quad = \quad \sum_{i=5}^{2n} 3i - 10(2n+4))$

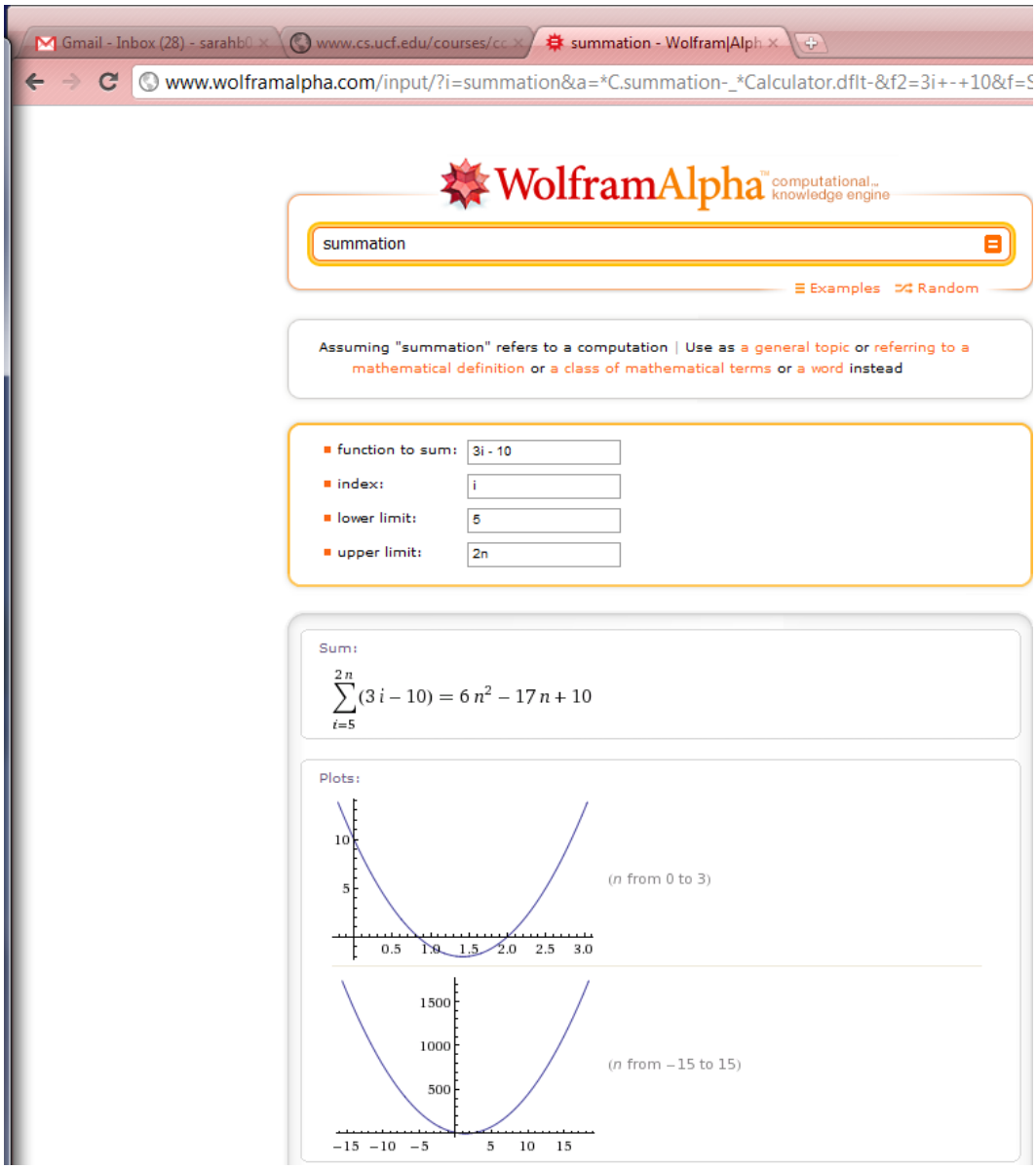
$= 3[\sum_{i=1}^{2n} i - \sum_{i=1}^{4} i] - 20n - 40 = 3[\frac{2n(2n+1)}{2} + \frac{4*5}{2}] - 20n - 40$

$= 3(2n^2 + n + 10) - 20n - 40 = 6n^2 + 3n + 30 - 20n - 40$

$= 6n^2 - 17n - 10$

**Closed form solution:** $\mathbf{6n^2 - 17n - 10}$

You can check your summation answers at: [http://www.wolframalpha.com](http://www.wolframalpha.com)




3) (10 pts) Write a **recursive** function that searches for a given value in an **unsorted** integer array. Return 1 if the value is found, 0 otherwise. The three input values to the function are the array, length of the array, and the value for which to search. Fill in the prototype given below:

```
int search(int array[], int length, int value) {
     if (length >= 0) {
          if (array[length-1] == value)
               return 1;
          else
               return search(array, length-1, value);

     }
     return 0;
}
```

4) a) If a given $O(n^3)$ algorithm runs in 12 ms for an input of size 25, and for another input runs in 324 ms, how big was that input, in all likelyhood?

$c * n^3 = time$

$c* 25^3 = 12$ ms

$c = 12/25^3$

Now time = 324ms, n = ?

$12/25^3 * n^3 = 324 \rightarrow n^3 = 324 * 25^3/12 \rightarrow n^3 = 2^2 3^4 * 25^3/2^2 3 \rightarrow n^3 = 3^3 * 25^3 \rightarrow$

**n = 75**

b) If a given $O(n \log_2 n)$ algorithm runs in 24 ms for an input of size 16, what would the expected run-time be for an input size of 64?

$c * n \log_2 n = time \rightarrow c * 16 \log_2 16 = 24ms \rightarrow c* 16 * 4 = 24ms \rightarrow c = 3/8$

Now n = 64, time = ?

$3/8 * 64 \log_2 64 = 24 \log_2 64 = 24 * 6 =$

**144ms**

c) Determine the Big-O runtime of the following segment of code:

```
int func4(int** array, int n)  {

    int i=0, j;
    while (i < n) {
        while (j < n && array[i][j] %2 == 0)
             j++;
        i++;
    }
    return j;
}
```

Since j never gets re-set after the first iteration of the inner loop, the algorithm only loops 2n times total → n times for the inner loop + n times for the outer loop. Giving us:
**O(n)**

5) (12 pts) The following struct can be used to store information about a student:

```
struct student {
  char first[20];
  char last[20];
  int* grades;
};
```

The last item of the struct is meant to be a pointer to a dynamically allocated array of ints that stores test scores.

Write a segment of code that asks the user to enter two positive integers:

1) n, representing the number of students
2) m, representing the number of tests each student takes.

and allocates space for a dynamically allocated array of size n of struct student, where each student has space to store exactly m test scores.

Note: You don't need to initialize any of the allocated space, you just need to allocate it.

The beginning part of the code segment is given to you. Declare any extra variables you need.

```
int n,m;
struct student* studList;
printf("How many students do you have?\n");
scanf("%d", &n);
printf("How many tests have they taken?\n");
scanf("%d", &m);
```

```
studList = (struct student*)malloc(sizeof(struct student) * n);
int i = 0;
for (i; i < n; i++)
    studList[i].grades= (int*)malloc(sizeof(int) * m);
```

6) (10 pts) Free the memory allocated to `studList` in the previous question.

```
int i;
for(i=0; i<n; i++)
     free(studList[i].grades);
free(studList);
```

7) (10 pts) Write a **recursive** function that prints out the items in a linked list in reverse order. The struct and function prototype are given below:

```
struct ll {
  int data;
  struct ll* next;
};

void print(struct ll* list) {

  if(list!= NULL)
  {
     print(list->next);
     printf("%d \n", list->data);
  }

}
```

8) (10 pts) Write a function that takes in a pointer to a linked list and an integer that represents a threshold value, and returns the number of values in the linked list greater than the given threshold value. Fill in the prototype given below:

```
int numAboveVal(struct ll* list, int threshold) {

     int count=0;
     while (list!=NULL)
     {
         if (list ->data==threshold)
         {
             count++;
         }
         list = list ->next;
     }
     return count;



}
```

9)

Given the following variables:

```
int *dynArray;
int size = 10;
```

a) Dynamically allocate memory for the variable dynArray so that it can store an array of size integers.

```
dynArray= (int *)malloc (size*sizeof(int));
```

b) Now expand dynArray to be able to hold 3 times the amount of integers than it does now:

```
dynArray= (int *)realloc (dynArray, (3*size*sizeof(int));
```

10)

Fill in the table below to show what the High, Low, and Mid indices for each iteration of Binary Search would be when searching the following array for value 32:

| Index: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Value: | 9 | 13 | 14 | 15 | 16 | 30 | 50 | 55 | 64 |

| **Low** | **High** | **Mid** |
|---|---|---|
| **0** | **8** | **4** |
| **5** | **8** | **6** |
| **5** | **5** | **5** |
| **6** | **5** | Stops |

The algorithm terminates when Low > High.