24/75

I pledge to take this assessment with honor and integrity and to only submit work that is my own. I will not share the contents of this assessment with my peers.

Dr. Nevard
September 11, 2023

Name: Tim Rokhud
Data Structures

4

1. The intent of this problem is to actually figure out what the functions do, not give a long boring incomprehensible translation of each line of code into English. My answers (which I presumably would give full credit to!) are each less than twelve words long! Each of the functions is shown on the included page of code.

(a) What does the function `x1(int n)` do? Here, $n > 1$.

This function prints all the n terms of the fibonacci sequence that are less than or eaual to n. ✓

(b) What does the function `x2(int a, int b)` do? Here, $a \geq 0$ and $b \geq 0$.

returns b + 2b + ... + a/2 b where n is a/2 (C++ integer division, i.e 5/2=2)

-2

(c) What does the function `x3(int n)` do? Here $n \geq 0$.

returns n.

-3

4

2. Write a function to evaluate a polynomial

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

using as few multiplications as you can. A correct implementation having only n multiplications gets full credit. (You may not use `pow()` or anything equivalent.).

```
double eval(double x, double a[], int n) {
    double total = 0;
    double xPow = 1;
    for(int i=0; i<n<++i) {
        total += a[i] . xPow;
        xPow = xPow . x;
    }
    return total;
```

2n mals

IGNORE

```
// not sure how to do power in O(1) time, so I will have
// many multiplications :( , returns n^x assuming x is int
double power (double n, double x) {
    double total = 1;
    while (x > 0) { total = total^2 . n; x--; }
    return total;
}
```

compare

a <= b

b > a

10

3. Given two sorted arrays a[] and b[], containing na and nb elements, respectively, write a function to merge the two arrays into one sorted array c[]. (You may assume that c has enough space for all the elements of a[] and b[]:

```
void merge(double c[], double a[], double b[], int na, int nb) {
    int ai=0, bi= 0; ci =0;

    while( (ai< na) && (bi<nb)) {
        if(a[ai] > b[bi]){
            c[ci++] = b[bi++];
        }
        else {
            c[ci++] =  a[ai++];
        }
    }
while ( ai < na ){
    c[ci++] =  a[ai++];
}
while ( bi < nb){
    c[ bi++] = b[bi++];
}

}
```

4. We have n disks of different sizes. Each disk has a small hole in its center so that it will fit on any of three pegs, which we will call the left, middle and right pegs. Initially, all n disks are on the left peg and we want to transfer all of them to the right peg. The largest disk is on the bottom, the second largest is next, and so on up to the smallest disk on the top. Now, the rules for transferring disks are as follows: Each disk must be transferred one at a time, no disk may ever be on top of a smaller disk and *disks may only be transferred between adjacent pegs, i.e., left to middle, middle to right, right to middle and middle to left.*

   (a) If T_n is the minimum number of transfers required to move all n disks from the left peg to the right, write a recurrence relation satisfied by T_n.

   (b) Solve the recurrence and use induction to prove your answer correct.

(c) Write a recursive routine to print out the sequence of moves necessary to transfer the n disks (You can use the following to indicate the transfer of a disk:

```
cout << "Transfer disk from " << a << " to " << b << endl; )
```

```
void f(int n, int from, int mid, int to) {
```

}

5. Let $f(N)$ be the `int` returned by the routine `mul()` shown on the included page of code. Determine $f(N)$.

f(n) = $\sum$

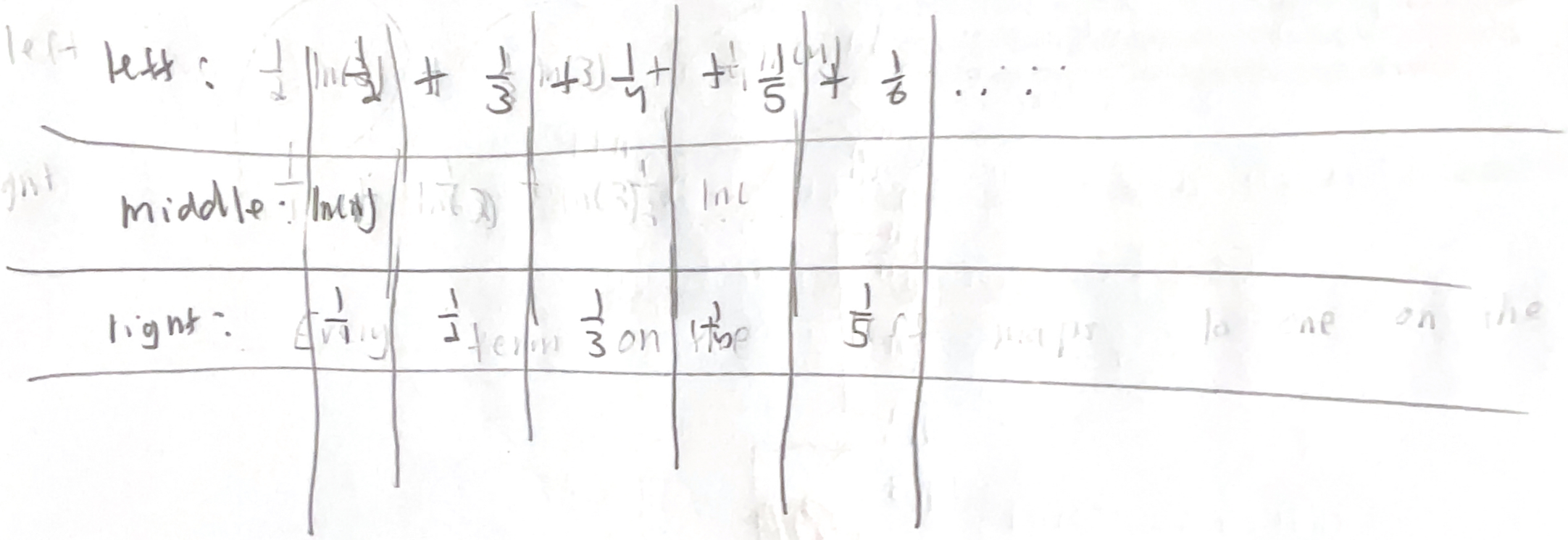
some sort of summation by the size of the
array by n.

not sure

(2)

6. Prove (a convincing explanation, perhaps using a picture, could suffice): For $n \geq 2$:

$$\sum_{k=2}^{n} \frac{1}{k} < \ln n < \sum_{k=1}^{n-1} \frac{1}{k}$$



left will always be less than the right

right is a summation that increases at a decreasing rate, but it begins at 1, making it more than ln(n)

left begins at $\frac{1}{2}$ and has a smaller end value of $\frac{1}{n}$, making it smaller than ln(n)

7. Consider functions from the positive reals into the positive reals. If we have two such functions, say f and g, we say that $f(n) = O(g(n))$ if there is $M > 0$ such that $f(n) < Mg(n)$ for all $n > 0$. Arrange the following functions so that if $f(n) = O(g(n))$ and $g(n) = O(f(n))$ then f and g appear on the same line, but if $f(n) = O(g(n))$ and the second equation doesn't hold, g must appear higher on the page than f. The functions are (lg denotes the base-2 logarithm):

2^n $\sqrt{2}^{\lg n}$ $n!$ n^3 2^{2^n} $n^{\frac{1}{\lg n}}$ $\lg \lg n$ $n^{\lg \lg n}$ e^n $n \lg n$ $\sqrt{\lg n}$ $2^{2^{n+1}}$