Exam #2

Thursday, September 5, 2019

- This exam has 6 questions, with 100 points total.
- You have two hours.
- **You should submit your answers to the corresponding places in the exam on the NYU Classes system.**
- In total, you should upload 3 '.cpp' files:
  - One '.cpp' file for questions 1-4.
    Write your answer as one long comment (/* … */).
    Name this file 'YourNetID_q1to4.cpp'.
  - One '.cpp' file for question 5, containing your code for section (a), and the answer to section (b) typed as a comment.
    Name this file 'YourNetID_q5.cpp'.
  - One '.cpp' file for question 6, containing your code.
    Name this file 'YourNetID_q6.cpp'.
- **Write your name, and netID at the head of each file.**
- This is a closed-book exam. However, you are allowed to use Xcode or Visual-Studio.
  You should create a new project and work ONLY in it.
  You may also use two sheets of scratch paper.
  Besides that, no additional resources (of any form) are allowed.
- Calculators are **not** allowed.
- Read every question completely before answering it.
  Note that there are 2 programming problems at the end**. Be sure to allow enough time for these questions**

Part I – Theoretical:

- *You should submit your answers to all questions in this part (questions 1-4) in **one** '.cpp' file. Write your answers as one long comment (`/* ... */`).*
  Name this file 'YourNetID_q1to4.cpp'.

- *For questions in this part, try to find a way to use regular symbols.*
  For example, instead of writing $\Theta(n)$, you could write theta(n),
  instead of writing $\binom{n}{k}$, you could write C(n, k), etc.
  Alternatively, you could also make a note, at the beginning of your answer, stating which symbol you used to indicate a specific mathematical notation.

Question 1 (14 points)

Use mathematical induction to prove that for every positive integer $n \geq 4$, $n! \geq 2^n$.

Notes:

1. The *factorial function* is defined for any integer $n \geq 1$ as the product of all the positive integers through n: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$
2. A non-inductive proof will not get any points.

Question 2 (12 points)

Three student council representatives are chosen from a group of 8 girls and 4 boys.
In how many choices there are more girls than boys?
Explain your answer.

Question 3 (15 points)

A die has the numbers 1, 2, 2, 3, 3, 3 on its six sides.
Let *X* be a random variable equal to the number showed when this die is rolled.

a. Find the distribution of *X*
b. Find the expected value of *X*

Explain your answers.

Question 4 (14 points)
Analyze its running time of `func1` and `func2`.
Explain your answers.

Note: Give your answers in terms of asymptotic order. That is, $T(n) = \Theta(n^2)$, or $T(n) = \Theta(\sqrt{n})$, etc.

```
int func1(int n){
    int i, j;
    int count;

    count = 0;
    for (i = 1; i+i <= n; i++)
        for (j = 1; j <= i; j++)
            count++;
    return count;
}


int func2(int n){
    int i, j;
    int count;

    count = 0;
    for (i = 1; i*i <= n; i++)
        for (j = 1; j <= i; j++)
            count++;
    return count;
}
```

Part II – Coding:

- *Each question in this part (questions 5-6), should be submitted as a '.cpp' file.*
- *Pay special attention to the style of your code. Indent your code correctly, choose meaningful names for your variables, define constants where needed, choose most suitable control statements, etc.*
- *In all questions, you may assume that the user enters inputs as they are asked. For example, if the program expects a positive integer, you may assume that user will enter positive integers.*
- *No need to document your code. However, you may add comments if you think they are needed for clarity.*

Question 5 (30 points)

In this question we represent a positive integer in its base-2 representation, and store it in an array of `int`s. The elements of the array would be the bits of the number, in the order from most-significant to least-significant.
For example, since $38 = (100110)_2$, 38 would be represented with the following 6-element array: [1, 0, 0, 1, 1, 0].

a. Implement the following function:

```
void makeBase2(int num, int*& outBitsArr, int* outBitsArrSize)
```

When this function is called with a positive integer `num`, it should create an array, containing the bits of its base-2 representation, as described above.
This array should be returned by using the two output parameters: `outBitsArr` and `outBitsArrSize` to update the base address of the created array, and its logical size. Note that these output parameters use different methods to update the values in the scope of the caller of the function. `outBitsArr` uses call by reference, where `outBitsArrSize` uses a pointer.

For example, the call to `makeBase2` with `num=38`, should create the array `[1, 0, 0, 1, 1, 0]` and use the output parameters to return its base address and its logical size.

Implementation requirements:

1. You must implement the function with the prototype as given above. That is, you are not allowed to change the header line of the function.
2. Aim for the best asymptotic runtime.

b. You are given the following program, that calls the function described in section (a):

```
1.  int main(){
2.      int* bitsArr;
3.      int bitsArrSize;
4.
5.      makeBase2(38, ______, ______);
6.
7.      cout<<"bitsArr:  ";
8.      printArray(bitsArr, bitsArrSize);
9.
10.     delete []bitsArr;
11.     return 0;
12. }
13.
14. void printArray(int arr[], int arrSize){
15.     for(int i = 0; i < arrSize; i++)
16.         cout<<arr[i]<<" ";
17.     cout<<endl;
18. }
```

Complete line 5 of the program above, so when executed, it would print:

```
bitsArr: 1 0 0 1 1 0
```

Question 6 (15 points)

A ***geometric progression*** is a sequence of numbers where each term after the first is found by multiplying the previous one by a fixed, non-zero number, called the *common ratio*.
For example, the sequence: *2, 6, 18, 54, 162, ...* is a geometric progression (with common ratio *3*).
Similarly, *10, 5, 2.5, 1.25, ...* is a geometric progression (with common ratio *½*).

Give a **recursive** implementation for:

```
bool is_geometric_progression(double* seq, int n)
```

The function is given `seq`, a base address of an array containing numbers, and its logical size `n`.
When called, it should return `true` if `seq` contains the first `n` elements of a valid geometric progression, or `false` otherwise.

For example, if `seq = [2.0, 6.0, 18.0, 54.0, 162.0]`,
the call `is_geometric_progression(seq, 5)` should return `true`.

Implementation requirements:

- Your function should run in **worst case linear time**. That is, it should run in $\theta(n)$.
- Your function **must be recursive**.

Note: You don't need to write a `main()` program.