CS340 – Advanced Data Structures and Algorithm Design – Fall 2020
Assignment 1 – September 11, 2020

Dr. Sandra Zilles, Department of Computer Science, University of Regina

due September 21, 2020, 10.00 am

- Please submit a single pdf (can include scans/photographs of your handwriting) including code listings and screenshots/listings of results from running your code. The single pdf must be submitted through UR Courses.
- Please submit, in a separate file, your C++ code (compilable in Visual Studio or g++) through UR Courses.

Problem 1 (1+1+1+1 marks). For each of the following functions T, determine the simplest possible function f such that $T(N) = \Theta(f(N))$. Show the calculation that simplifies $T(N)$ if necessary. You do not have to prove that $T(N) = \Theta(f(N))$, i.e., you do not need to give the constants c and n_0.

(a) $T(N) = 5.3 \cdot N^2 \cdot N\sqrt{N} + 22N^3 \cdot \log(N) + 100N\sqrt{N}$

(b) $T(N) = \frac{(96\log(N)\log(N)+500\cdot N^2)\cdot 0.1}{4N}$

(c) $T(N) = \frac{0.11\cdot N\cdot \log^4(N)+5N}{\log(N)}$

(d) $T(N) = \frac{3}{N^2} \cdot \log(N) \cdot (12 \cdot \log(N) + \cdot 3^N + 19)$

Problem 2 (3+3 marks). (a) Let $T : \mathbb{N} \to \mathbb{R}^{\geq 0}$ and $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ be any two functions. Prove that $[T(N) = o(f(N))$ and $f(N) = \Theta(g(N))]$ implies $T(N) = o(g(N))$. Hint: If you use definitions from class and the argument we used to explain why $0.05N^2 + 1000N = O(N^3)$ (though that's not the best possible Big-O bound), then you do not need to use constants c or n_0 in your argument.

(b) Provide functions $T : \mathbb{N} \to \mathbb{R}^{\geq 0}$, $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$, and $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ such that $T(N) = O(g(N))$, $f(N) = o(g(N))$, and $T(N) \neq \Omega(f(N))$ simultaneously. Explain your choice (without proof).

Problem 3 (3+3+3 marks). For each of the following code fragments, determine the best possible asymptotic upper bound on its running time, depending on `n`. Give a brief explanation for each of your answers.

(a)
```
for (i = n; i > 0; i--)
   for (j = 0; j < n; j=j+2)
      cout << "It's not winter yet." << endl;
```

(b)
```
for (i = 1; i < n; i=2*i)
   for (j = 0; j < floor(n/2); j++)
      cout << "And even though winter is bound to come, it won't
      last forever." << endl;
```

(c)
```
int myFunction(int n)
{
   if (n<=1)
      return 1;
   else
      return myFunction(n-1)*n;
}
```

Problem 4 (4+2+4 marks). (a) Write a C++ program according to the following requirements.

- Your main program expects at least one integer value `n` as its input (in order to later evaluate `assignment1Algorithm` on input `n`). You can either let your program accept more than one input at once or you can just call it several times if you want to observe the behaviour of `assignment1Algorithm(n)` for several values of `n`. Input values can be passed when calling the program or can be prompted by the program, as you prefer.
- Your main program declares variables named `start` and `finish` of type `clock_t`.
- In order to evaluate `assignment1Algorithm` on input `n`, your main program processes the following fragment.

```
start = clock();
result = assignment1Algorithm(n);
finish = clock();
timeUsed = ((long double)(finish - start))/CLOCKS_PER_SEC;
```

  This essentially stores the total time consumed by `assignment1Algorithm(n)` in `timeUsed`.
- Your program outputs the following values (via `std::cout`) for every input value `n`:
  (i) the result returned by `assignment1Algorithm(n)`, stored in a variable named `result`,
  (ii) the value of `n` divided by `result`,
  (iii) the value of `n*log2(n)` divided by `result`,
  (iv) the value of `pow(n,1.5)` divided by `result`,
  (v) the total time `timeUsed` used by `assignment1Algorithm` on input `n`,[1]
  (vi) the value of `n` divided by `timeUsed`,[1]
  (vii) the value of `n*log2(n)` divided by `timeUsed`,[1]
  (viii) the value of `pow(n,1.5)` divided by `timeUsed`.[1]

  [1]You may multiply the values (v) through (viii) by a constant factor (like 10^5 or 10^6) to be able to better observe the differences between the numbers you record.

In this problem, the C++ code for `assignment1Algorithm` is the following:

```
long long assignment1Algorithm(long long n)
{
    long long sum = 0;

    for(long long i=1; i<=n; i++)
    {
        for(long long j=n; j>1; j=floor(j/2))
        {
            sum += 1;
        }
    }
    return(sum);
}
```

(b) Report the output of your program on the following inputs for `n`: 10, 100, 1,000, 10,000, 100,000, 1,000,000, 10,000,000, 20,000,000, 50,000,000, 100,000,000, 300,000,000.

(c) Taking the outputs reported in (b) into account, what do you conclude concerning the asymptotic growth of the result returned by `assignment1Algorithm(n)` for growing `n`? What do you conclude concerning the asymptotic growth of the running time of `assignment1Algorithm(n)` for growing `n`? What do you conclude about the growth rates of the three functions $f_1(n) = n$, $f_2(n) = n \cdot \log(n)$, and $f_3(n) = n\sqrt{n}$, compared to one another?