Lab01-Preliminary

VE281 - Data Structures and Algorithms, Xiaofeng Gao, TA: Qingmin Liu, Autumn 2019

∗ Please upload your assignment to website. Contact webmaster for any questions.

∗ Name:Wu Jiayao   Student ID:517370910257   Email: jiayaowu1999@sjtu.edu.cn

1. What is the time complexity of the following code?

```
1  // REQUIRES: an integer k
2  // EFFECTS: return the number of times that Line 12 is executed
3  int count(int k)
4  {
5      int count = 0;
6      int n = pow(2,k); // n=2^k
7      while (n>=1)
8      {
9          int j;
10         for (j=0;j<n;j++)
11         {
12             count += 1;
13         }
14         n /= 2;
15     }
16     return count;
17 }
```

Solution.

$$2^k + 2^{k-1} + 2^{k-2} + ... + 1 = 2^{k+1} - 1 = O(2^k).$$ So the time complexity is $O(2^n)$.

□

2. Given an array **nums** of n integers, are there elements a, b, c in nums such that $a+b+c=0$? Write a program to find all unique triplets in the array which gives the sum of zero. Give your code as the answer. **Claim that the time complexity of your program should be less than or equal to $O(n^2)$.**

Examples: Input array [-1, 0, 1, 2, -1, -4], the solution is [[-1, 0, 1], [-1, -1, 2]]

Solution. Please explain your design and fill in the following block:

```
// REQUIRES: an integer array a of size n
// EFFECTS: return a list of triplets, the sum of each triplet equals to 0.
vector<vector<int>> threeSum(vector<int>& a) {
        vector<vector<int>> res;
        std::sort(a.begin(),a.end());
        for (int i = 0 ; i< a.size() ; i++)
        {
            int target = -a[i];
            int front = i + 1;
            int back = a.size() - 1;
```

```
            while (front < back)
            {
                int sum = a[front] + a[back];
                if (sum > target)
                    back --;
                else if (sum < target)
                    front ++;
                else
                {
                    vector<int> tmp (3,0);
                    tmp[0] = a[i];
                    tmp[1] = a[front];
                    tmp[2] = a[back];
                    res.push_back(tmp);

                    while (front < back && a[front] == tmp[1])
                        front ++;
                    while(front < back && a[back] == tmp[2])
                        back--;
                }
            }
            while(i+1 < a.size()-1 && a[i+1]==a[i])
                i++;
        }
        return res;
    }
```

Explain the time complexity of your solution here.

$$\text{The time complexity for std::sort is: } O(n \cdot \log n).$$

$$\text{The time complexity is: } \Sigma_{i=1}^{n-2} n - i = \frac{1}{2}(n^2 - 3n + 2) = O(n^2);$$

$$\text{The overall time complexity is: } O(n^2);$$

□

3. Equivalence Class

Definition 1 (o-Notation)**.** *Let $f(n)$ and $g(n)$ be functions from the set of natural numbers to the set of nonnegative real numbers. $f(n)$ is said to be $o(g(n))$, written as $f(n) = o(g(n))$, if*

$$\forall c. \exists n_0. \forall n \geq n_0. f(n) < cg(n).$$

An equivalence relation $\mathcal{R}$ on the set of complexity functions is defined as follows:

$$f\mathcal{R}g \text{ if and only if } f(n) = \Theta(g(n)).$$

A complexity class is an equivalence class of $\mathcal{R}$.

The equivalence classes can be ordered by $\prec$ defined as: $f \prec g$ iff $f(n) = o(g(n))$.

Example: $1 \prec \log\log n \prec \log n \prec \sqrt{n} \prec n^{\frac{3}{4}} \prec n \prec n\log n \prec n^2 \prec 2^n \prec n! \prec 2^{n^2}$.

Please order the following functions by $\prec$ and give your explanation:

$$(\sqrt{2})^{\log n}, (n+1)!, ne^n, (\log n)!, n^3, n^{1/\log n}.$$

Solution.

$$n^{1/\log(n)} \prec (\sqrt{2})^{\log n} \prec n^3 \prec (\log n)! \prec ne^n \prec (n+1)!$$

Since

$$(\sqrt{2})^{\log n} = n^{1/2} \prec n^3$$
$$n^{1/\log n} = 2$$

We can get:

$$n \cdot n^2 \prec n \cdot 2^n \prec n \cdot e^n \prec (n+1) \cdot n!,$$

which leads to

$$n^{1/\log n} \prec (\sqrt{2})^{\log n} \prec n^3 \prec ne^n \prec (n+1)!.$$

According to Stirling's Approximation,

$$(\log n)! \leq 2(\log n)^{1/2+\log n} \times 2^{-\log n}$$

$$(\log n)^{\log n} = 2^{(\log n)\times(\log\log n)}$$

So the approximate growth is $n^{(\log\log n)-1}$. Obviously, $n^3 \prec n^{(\log\log n)-1}$. Hence

$$n^3 \prec (\log n)!$$

Since $n! \prec 2^{n^2}$,

$$(\log n)! \prec 2^{(\log n)^2} = n \cdot n \prec n2^n \prec ne^n$$

$\square$