Home work 3

Bài 1:

1. Bài 206 trang 37.

$$\begin{cases} T(1) = 1 \\ T(n) = 2T(n-1) + n - 1, \ \forall n \geq 2 \end{cases}$$

Ta có $T(n) = 2\,T(n-1) + n - 1 = 2\{2\,T(n-2) + (n-1-1)\} + n - 1$

$= 4\,T(n-2) + 2(n-2) + (n-1)$

$= 2^2 T(n-2) + 2^1(n-2) + 2^0(n-1)$

$= 2^2\{2\,T(n-3) + (n-2-1)\} + 2^1(n-2) + 2^0(n-1)$

$= 2^3\,T(n-3) + 2^2(n-3) + 2^1(n-2) + 2^0(n-1)$

$= \ldots.$

$= 2^i\,T(n-i) + \sum_{j=0}^{i} 2^{i-1}(n-j)$

$= \ldots.$

$= 2^{n-1}\,T(n-(n-1)) + \sum_{j=0}^{n-1} 2^{j-1} * (n-j) = 2^{n-1}\,T(1) + \sum_{j=0}^{n-1} 2^{j-1} * (n-j)$

Ta có: $\lim_{n\to\infty} \frac{T(n)}{2^n} = \lim_{n\to\infty} (\frac{1}{n} + \sum_{j=0}^{n-1} 2^{j-1-n}(n-j)) = 0$

Vậy $T(n) = O(2^n)$

2. Bài 224 trang 38

Ta có: $X(2) = 1 = 2^0$ thỏa mãn $X(n) \geq 2^{n-2}$

$X(3) = 2 = 2^1$ thỏa mãn $X(n) \geq 2^{n-2}$

Giả sử bài toán đúng với mọi $n \leq k$. Tức là $X(k) \geq 2^{k-2}$. Ta sẽ chứng minh $X(k+1) \geq 2^{k-1}$

Thật vậy. Xét $A_i(k)$ là một cách biểu diễn 1 cách đặt dấu ngoặc để thực hiện tính tích của dãy k phần tử .

Để tính kết quả của $A_i(k)$ ta luôn phải xuất phát từ việc tính tích của dãy con (xx). Do đó với mọi cách đặt các dấu ngoặc để thực hiện tính tích dãy có k thừa số luôn xuất hiện dãy con (xx). Do phép nhân có tính kết hợp nên ta có thể thêm 1 thừa số x vào dãy con (xx) để tạo thành dãy con mới (xxx). Khi đó

$A_i(k)$ là dãy có $k + 1$ thừa số. Để tính tích của dãy mới ta cần tính tích của dãy con (xxx) đầu tiên. Có 2 cách tính tích dãy (xxx) là x(xx) và (xx)x.

Vậy với mỗi cách tính tích của dãy có k thừa số bất kì ta luôn tạo được ít nhất 2 cách tính của dãy có $k + 1$ thừa số. Mà số cách tính của dãy có k thừa số là $X(k) \geq 2^{k-2}$. Do đó $X(k + 1) \geq 2^{k-1}$

Vậy $\forall n \geq 2$ thì $X(n) \geq 2^{n-2}$

3. Bài tập 231 trang 29
   Ta có:

$$\begin{cases} T(1) = d \\ T(n) = aT(n/c) + b \quad \forall n \geq 2 \end{cases}$$

Theo định lý chính thì độ phức tạp của thuật toán trên là:

$$T(n) = \begin{cases} O(1) & ,khi\ a < 1 \\ O(log_c n) & ,khi\ a = 1 \\ O(n^{log_c(a)}) & ,khi\ a > 1 \end{cases}$$

- Chứng minh.

Ta có $T(n) = aT(n/c) + b$
$= a\,(aT(n/c^2) + b) + b$
$= a^2T(n/c^2) + ab + b$
$= \ldots.$
$= a^{log_c n}\,T(n/c^{log_c(n)}) + \sum_{i=0}^{log_c n - 1} a^i b$
$= n^{log_c a}\,T(1) + b\sum_{i=0}^{log_c n - 1} a^i$
$= d * n^{log_c a} + b * \sum_{i=0}^{log_c n - 1} a^i$

- Với $a < 1$ thì $\Rightarrow log_c a < 0 \Rightarrow \lim_{n \to \infty} n^{log_c a} = 0$ và $\sum_{i=0}^{log_c n - 1} a^i = hằng\ số.$
  $\Rightarrow T(n) = b*$ hằn số = hằng số. Hay $T(n) = O(1)$
- Với $a = 1 \Rightarrow log_c a = 0 \Rightarrow T(n) = 1 + b * log_c(n - 1)$
  $\Rightarrow T(n) = O(log_c n)$
- Với $a > 1$ thì $\Rightarrow log_c a > 0$ Khi đó $\lim_{n \to \infty} \frac{n^{log_c a}}{\sum_{i=0}^{log_c n - 1} a^i} = \infty$
  $\Rightarrow T(n) = O(n^{log_c a})$

Bài tập 2

1. Tìm số fibonacci thứ n.

- File code fibonacci.py
- Đánh giá thuật toán

Ta có: $\begin{cases} T(1) = 1 \\ T(2) = 2 \\ T(n) = T(n-1) + T(n-2) + 1 \end{cases}$

Ta thấy để tính T(n) cần 2 lần đệ quy, sau mỗi bước ta giảm n đi 1 nên T(n) có độ phức tạp của 1 hàm mũ. Tức là T(n) < a c^n với a, c là 1 hằng số. Ta sẽ đi tìm hằng số c

$T(n) = T(n-1) + T(n-2) \leq a\,c^{n-1} + a\,c^{n-2} \leq a * c^n$

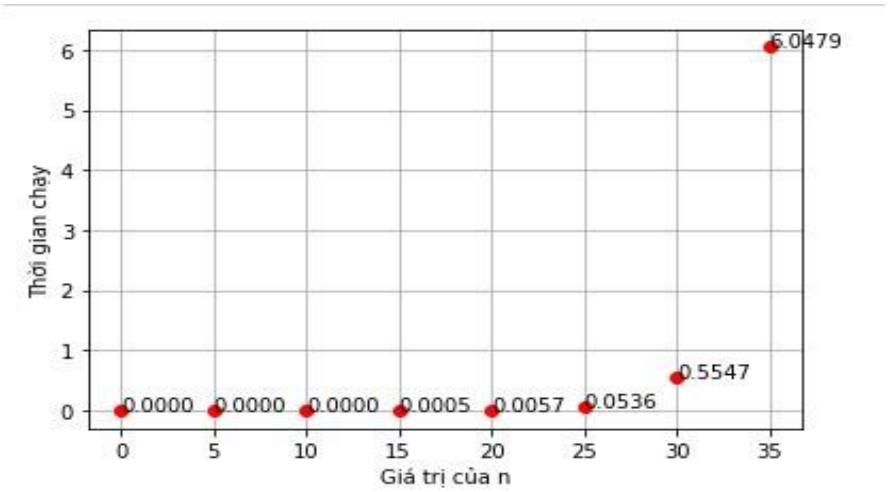
$=> c^n \geq c^{n-1} + c^{n-2}$

Chia cả 2 vế cho c^{n-2} $=> c^2 \geq c + 1$

Giá trị nhỏ nhất của c thỏa mãn bất phương trình trên là $\frac{\sqrt{5}+1}{2} \approx 1{,}618$

$=> T(n) = O(1{,}618^n)$

Chạy thuật toán bằng python 3



Ta thấy $\frac{t(35)}{t(30)} \approx \frac{t(30)}{t(25)} \approx \frac{t(25)}{t(20)} \approx 1{,}618^5 \approx 11$. Vậy $T(n) = O(1.618^n)$

2. Bài toán tháp Hà Nội
   - File code HaNoiTower.py

- Đánh giá thuật toán.

Ta có $T(0) = 0 = 2^0 - 1$

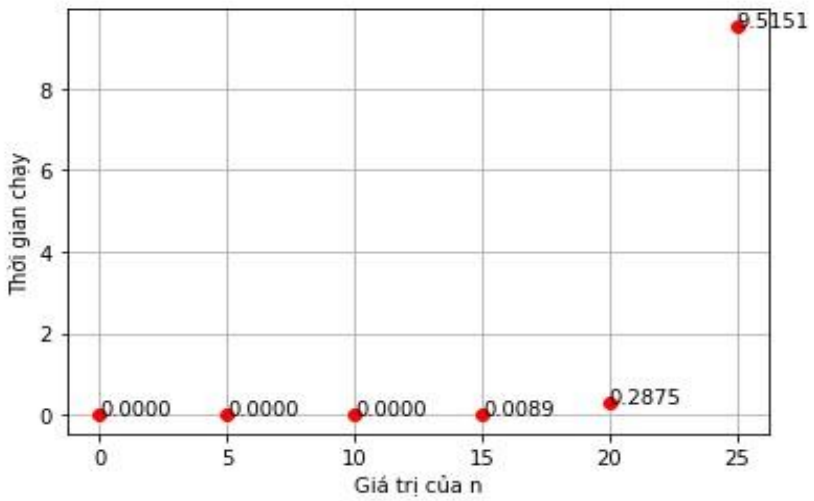
Giả sử $T(n-1) = 2^{n-1} - 1$

$T(n) = 2\,T(n-1) + 1$

$T(n) = 2 * 2^{n-1} - 2 + 1 = 2^n - 1$

Vậy thuật toán có độ phức tạp $O(2^n)$

Đồ thị thời gian chạy theo n viết bằng Python 3, thời gian tính bằng giây



Ta thấy $\frac{t(25)}{t(20)} \approx \frac{t(20)}{t(15)} \approx 32 = 2^5$. Vậy $T(n) = O(2^n)$

<u>Bài tập 3:</u>

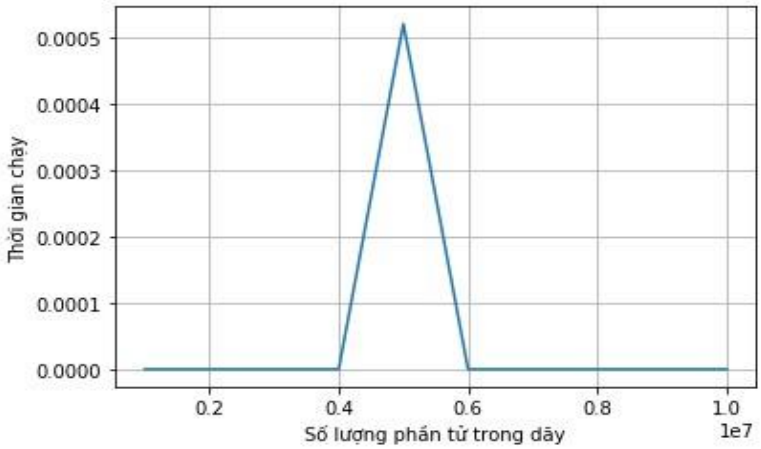
1. Tìm kiếm trong mảng đã được sắp xếp.
   - File code binarySerach.py
   - Đánh giá thuật toán

     Ta có $T(n) = T(n / 2) + 1$

     $=> T(n) = T(1) + log_2 n$

     $=> T(n) = O(log_2 n)$

     Đồ thị thời gian chạy theo số lượng phần tử



Do log(n) rất nhỏ nên ta sẽ tìm phần tử đầu tiên của mảng để thuật toán rơi vào trường hợp xấu nhất. Nhưng với mảng có 10 triệu phần tử thì thời gian tìm kiếm của trường hợp xấu nhất vẫn gần như là tức thời .

2. Tìm kiếm Min, Max trên mảng
    - File code minMaxRecursive.py
    - Đánh giá thuật toán

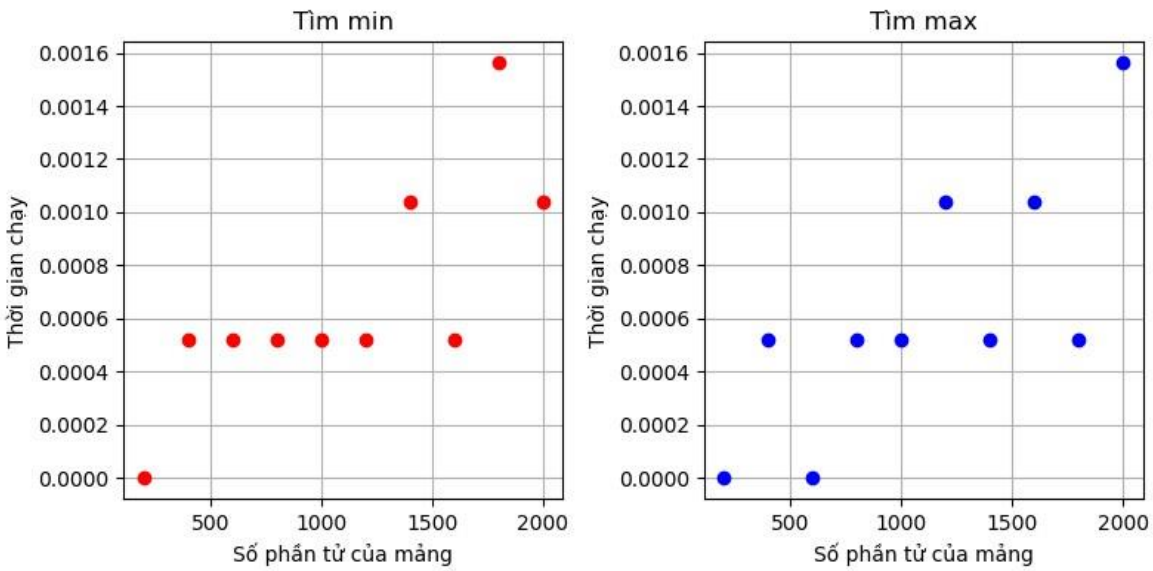
      Ta có $T(n) = T(n - 1) + 1$

      $T(n) = T(n - 2) + 2$

      $T(n) = \dots = T(1) + n - 1 = n$

      Vậy $T(n) = O(n)$

    Thời gian chạy của thuật toán theo sô lượng phần tử n



Dựa vào đồ thị ta thấy thời gian chạy có xu hướng tăng theo số phần tử của mảng theo quan hệ tuyên tính, tức là $T(n) = O(n)$

Bài tập 4:

- Bài toán chia thưởng: Có m vật thưởng được chia cho n học sinh giỏi có xếp hạng

theo thứ tự từ 1 đến n. Hỏi có bao nhiêu cách chia các phần thưởng thoả mãn các điều

kiện sau: (i) Học sinh giỏi hơn có số thưởng không ít hơn bạn kém hơn; (ii) m vật thưởng

phải chia hết cho các học sinh.

- Phân tích bài toán
  - Input: 2 số nguyên m, n là số lượng phần thưởng và số lượng học sinh cần chia phần thưởng
  - Output: số lượng cách chia phần thưởng thỏa mãn yêu cầu.
- Ý tưởng
  - Gọi $x_1, x_2 \ldots, x_n$ là số lượng phần thưởng tương ứng của học sinh xếp hạng từ 1 đến n.
  - Nếu $n = 1$ thì chỉ có 1 cách chia.
  - Nếu $m = 0$ thì chỉ có 1 cách chia: mọi học sinh đều có 0 phần thưởng.

- Nếu m = 1 thì chỉ có 1 cách chia: học sinh học giỏi nhất có 1 phần thưởng, mọi học sinh còn lại có 0 phần thưởng.
- Yêu cầu: $x_1 + x_2 + \dots + x_n = m$

$$0 \le x_n \le x_{n-1} \le x_{n-2} \dots \le x_1$$

- Do học sinh giỏi hơn có số thưởng không ít hơn bạn kém hơn nên số phần thưởng tối đa mà học sinh loại n (hạng thấp nhất) có thể nhận là $\left[\frac{m}{n}\right]$. Vậy số phần thưởng học sinh loại n nhận được nằm trong khoảng $\left[0, \left[\frac{m}{n}\right]\right]$.
- Giả sử số phần thưởng học sinh loại n nhận được là: x_n .Khi đó số phần thưởng còn lại chia cho n – 1 học sinh còn lại là : $m - \left[\frac{m}{n}\right]$. Bài toán trở thành chia $m - \left[\frac{m}{n}\right]$ phần thưởng cho n – 1 học sinh với yêu cầu:

$$x_1 + x_2 + \dots + x_{n-1} = m - \left[\frac{m}{n}\right]$$

$$x_n \le x_{n-2} \le x_{n-3} \dots \le x_1$$

- Với mỗi giá trị của X_n trong khoảng $\left[0, \left[\frac{m}{n}\right]\right]$ thì ta lại có 1 bài toán con là chia m - X_n phần thưởng cho n - 1 học sinh và mỗi học sinh nhận ít nhất X_n phần thưởng.
- Gọi S(m, n, 0) là số cách chia m phần thưởng cho n học sinh sao cho mỗi học sinh có ít nhất 0 phần thưởng

Khi đó: $S(m, n, 0) = \sum_{i=0}^{\left[\frac{m}{n}\right]} S(m - i,\ n - 1,\ i)$

- Thuật toán:

  Mã giả

```
program reward(m,n,x){
   temp = 0

   if(m == 0 || n == 1 || m == 1) return 1
   else{

      for( i = x to m/n){
         temp += reward(m - i, n - 1, i)
      }
   }
   return temp;
}
```

- Chứng minh tính đúng

  Ta thấy, $x \leq \frac{m}{n} = \frac{m\,(n-1)}{n(n-1)} = \frac{m-\frac{m}{n}}{n-1}$. Do đó $x \leq \frac{m}{n}$

  Xét $n = 1$, ta chỉ có 1 cách chia thưởng duy nhất, $S(m, n, x) = 1$. Do đó thuật toán đúng với $n = 1$.

  Giả sử thuật toán đúng với $n = k$. T sẽ chứng minh thuật toán đúng với $n = k + 1$.

  Ta có: $S(m, k+1, x)$ là số cách chia m phần thưởng cho $k + 1$ học sinh, mỗi học sinh nhận ít nhất x phần thưởng. Gọi i là số phần thưởng người thứ nhất nhận được, số cách chia thưởng thỏa mãn điều kiện là $S(m - i, k, i)$. Do mỗi người nhận số phần thưởng ít nhất là x và người xếp hạng ít nhất nhận số phần thưởng nhiều nhất là $m/(k+1)$ nên $i \in \left[x, \frac{m}{k+1}\right]$.

  Do đó

  $$S(m, k+1, x) = \sum_{i=x}^{\frac{m}{k+1}} S(m-i,\ k,\ i).$$

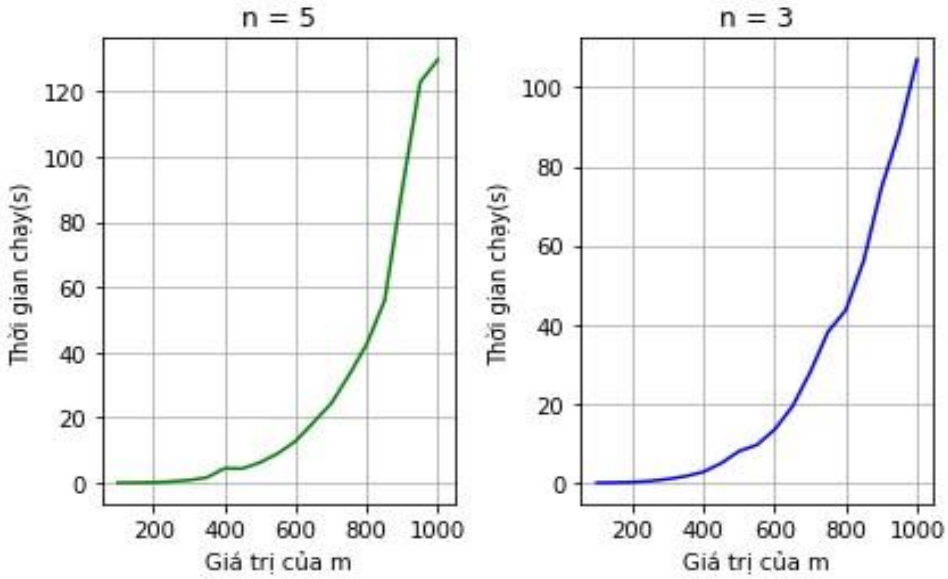
  mà $S(m - i, k, i)$ tính được kết quả đúng theo giả thiết. Vậy $S(m, k+1, x)$ cho ra kết quả đúng.

- File code reward.py
- Phân tích thuật toán
  - $T(m, 1) = 1$ là số cách chia m phần thưởng cho 1 học sinh

- $T(m, n, x) = \sum_{i=x}^{\frac{m}{n}} T(m-i, n-1, \ i) < \frac{m}{n} T(m, \ n-1, 0) = \left(\frac{m}{n}\right)^n$

Vậy thuật toán có độ phức tạp $O\left(\left(\frac{m}{n}\right)^n\right)$

Đồ thị thuật toán chạy bằng Python:



Ta thấy: Với cùng giá trị m, thì n càng lớn thì thời gian thực thi thuật toán càng lâu. Đồ thị có dạng của 1 hàm mũ.

- Một số file code có thời gian thực thi rất lớn do yêu cầu về kiểm tra thời gian chạy của thuạt toán.