Homework 2

@March 12, 2023

I. Xác định kích thước dữ liệu vào và bậc tăng trưởng độ phức tạp thuật toán

1. Xác định kích thước đầu vào, phép toán thực hiện và số lượng tính toán thay đổi với dữ liệu nhập vào khác nhau
    a. Trung bình cộng của n phần tử:
        - Kích cỡ: n
        - Phép toán: cộng và chia
        - Số lần thực hiện giống nhau đối với các đầu vào có cùng kích thước.
    b. Tính $n/n!$
        - Kích cỡ: n
        - Các phép toán: Nhân, chia và tính giai thừa
        - Số lần thực hiện sẽ khác nhau với đầu vào có cùng kích thước, vì hàm giai thừa tăng mạnh theo giá trị của n.
    c. Tìm phẩn tử nhỏ nhất trong danh sách n phần tử.
        - Kích cỡ: n
        - Phép toán: so sánh
        - Số lần thực hiện giống nhau đối với các đầu vào có cùng kích thước.
    d. In dãy n phần tử đảo ngược
        - Kích cỡ: n
        - Phép toán: in ra màn hình
        - Số lần thực hiện giống nhau đối với các đầu vào có cùng kích thước.
    e. Đảo ngược danh sách n phần tử
        - Kích cỡ: n
        - Phép toán: Hoán vị
        - Số lần thực hiện giống nhau đối với các đầu vào có cùng kích thước.
    f. Thuật toán pen-and-pencil để cộng 2 số nguyên có n chữ số.

- Kích cỡ: n
- Phép toán: Cộng, nhớ, trả.
- Số lần thực hiện giống nhau đối với các đầu vào có cùng kích thước.

3. Consider a classic sequential search that scans a list to search the occurrences of a given search key in the list. Discuss the worst case, average case, and best case efficiency of classic sequential search.

- Worst case: Độ phức tạp O(n) khi phải duyệt qua hết tất cả các phần tử trong danh sách.
- Best case: O(1): Phần tử cần tìm nằm ngay ở vị trí xuất phát.
- Average case: O(n/2) ~ O(n)

3. **Suggest how any sorting algorithm can be augmented in a way to make the best-case count of its key comparisons equal to just n − 1 (n is a list's size, of course). Do you think it would be a worthwhile addition to any sorting algorithm?**
    - Thuật toán sắp xếp chỉ cần thực hiện n-1 phép so sánh trong trường hợp tốt nhất: mảng đã được sắp xếp sẵn. Lúc đấy chỉ cần duyệt qua 1 lần với n-1 phép so sánh là đủ.
    - Nhưng trên thực tế, trường hợp tốt nhất thường không được quan tâm vì ít khi xuất hiện. Thay vào đó thì trường hợp trung bình và tệ nhất mới đại diện được cho độ hiệu quả
    - Vậy nên sẽ chả có mấy ai thêm 1 vòng for vào để kiểm tra cho trường hợp tốt nhất.

II. Xác định mối quan hệ độ phức tạp thuật toán theo kí pháp tiệm cận

2.3

For belongs to. (Use the simplest g(n) possible in your answers.) Prove your each of the following functions, indicate the class $(g(n))$ the function assertions.

a. $f(n) = (n^3 + 1)^6$ $g(n) = n^{18}$

$\lim \frac{f(n)}{g(n)} = \lim \frac{(n^3+1)^6}{n^{18}} = 1$

$\rightarrow f(n) \in \Theta(g(n)) = \Theta(n^{18})$

b. $f(n) = \sqrt{10n^4 + 7n^2 + 3n}$ $g(n) = n^2$

$\lim \frac{f(n)}{g(n)} = \lim \frac{\sqrt{10n^4+7n^2+3n}}{n^2} = \sqrt{10}$

$\rightarrow f(n) \in \Theta(g(n)) = \Theta(n^2)$

c. $f(n) = 2n \log(2n + 2)^3 + (n^2 + 2)^2 \log n = 6n \log(2n + 2) + (n^2 + 2)^2 \log n$

$6n\log(2n+2) \in \Theta(n\log n)$

$(n^2+2)^2\log n \in \Theta(n^4\log n)$
$\to f(n) = \Theta(g(n)) = \Theta(3^n)$

e. $f(n) = 2\log_2 n\, g(n) = \log n$

$\lim \frac{f(n)}{g(n)} = \lim \frac{2\log_2 n}{\log n} = 2$

$\to f(n) \in \Theta(g(n)) = \Theta(\log n)$

2.5

- $(n^2+3)! \in \Theta(n^2!)$
- $2\log(n+50)^5 = 10\log(n+50) \in \Theta(\log n)$
- $3^{3n} = 27^n \in \Theta(27^n)$
- $0.05n^{10} + 3^{n3} + 1 = 0.05n^{10} + 27^n + 1 \in \Theta(27^n)$
- $\ln^3 n \in \Theta(\log^3 n)$
- $\sqrt{n} \in \Theta(\sqrt{n})$
- $3^{2n} = 9^n \in \Theta(9^n)$

Giờ sắp xếp theo thứ tự thấp đến cao:

$\Theta(\log n)$	$\Theta(\log^3 n)$	$\Theta(\sqrt{n})$	$\Theta(9^n)$	$\Theta(n^2!)$	$\Theta(27^n)$
$2\log(n+50)^5$	$\ln^3 n$	$\sqrt{n}$	3^{2n}	$(n^2+3)!$	$0.05n^{10} + 3^{n3} + 1$ và 3^{3n}

2.9 **We mentioned in this section that one can check whether all elements of an array are distinct by a two-part algorithm based on the array's presorting.**

a. **If the presorting is done by an algorithm with a time efficiency in $\Theta(n\log n)$, what will be a time-efficiency class of the entire algorithm?**

   Độ phức tạp vẫn là $\Theta(n\log n)$, vì việc sẽ chỉ mất thêm 1 lần duyệt ($O(n)$) để kiểm tra xem có phần tử nào giống nhau không.

b. **If the sorting algorithm used for presorting needs an extra array of size n, what will be the space-efficiency class of the entire algorithm?**

   Độ phức tạp không gian sẽ là $O(2n) \approx O(n)$

III. Thiết kế thuật toán, chứng minh tính đúng và xác định độ phức tạp của thuật toán

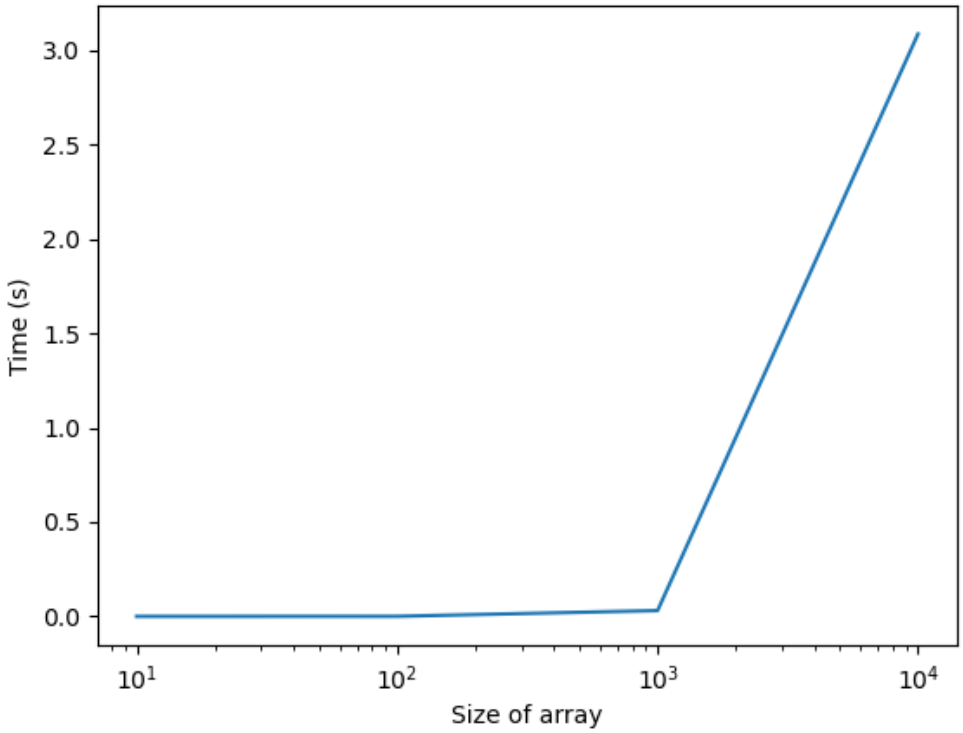
1. Mã giả:

```
procedure selectionSort(a: list of n elements)
  for i from 0 to n-2 do
          minIndex = i
          for j from i+1 to n-1 do
              if a[j] < a[minIndex] then
                  minIndex = j
              end if
          end for
          swap a[i] and a[minIndex]
      end for
end procedure
```

2. Chỉ cần thực hiện với n-1 phần tử đầu tiên, thay cho cả n phần tử, vì khi sắp xếp n-1 phần tử thì phần tử cuối cùng cũng sẽ được sắp xếp đúng vị trí, do đó không cần phải sắp xếp lại.
3. Đánh giá thời gian thực hiện thuật toán:
   - Trường hợp tốt nhất: Mảng đã được sắp xếp, vòng lặp trong thuật toán chỉ chạy qua mỗi phần tử một lần để tìm phần tử nhỏ nhất và không cần đổi chỗ, do đó độ phức tạp là $O(n^2)$.
   - Trường hợp xấu nhất: Mảng được sắp xếp ngược lại hoàn toàn, vòng lặp trong thuật toán sẽ chạy qua mỗi phần tử n-1 lần để tìm phần tử nhỏ nhất và đổi chỗ, do đó độ phức tạp là $O(n^2)$.
   - Code:

```
import random
import time
import matplotlib.pyplot as plt

def selection_sort(a):
    n = len(a)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if a[j] < a[min_index]:
                min_index = j
        a[i], a[min_index] = a[min_index], a[i]

# Sinh dãy số ngẫu nhiên với số lượng phần tử từ 10^1 đến 10^4
N = [10**i for i in range(1, 5)]
times = []

for n in N:
    a = [random.randint(0, 100) for _ in range(n)]
    start_time = time.time()
    selection_sort(a)
    end_time = time.time()
    times.append(end_time - start_time)

# Vẽ biểu đồ sự phụ thuộc thời gian vào kích thước n
plt.plot(N, times)
plt.xlabel('Size of array')
plt.ylabel('Time (s)')
```

```
plt.xscale('log')
plt.show()
```



Kết quả: Thuật toán sắp xếp chọn có độ phức tạp là O(n^2), nhưng thực tế nó có thể chạy nhanh hơn so với các thuật toán có độ phức tạp tương tự như vì số lần truy cập đến phần tử mảng là ít hơn. Tuy nhiên, đối với các dãy số lớn, độ phức tạp của thuật toán sẽ ảnh hưởng đến thời gian thực hiện.

- Bất biến của vòng lặp trong thuật toán sắp xếp chọn là sau khi thực hiện i vòng lặp, thì i phần tử đầu tiên trong mảng đã được sắp xếp theo thứ tự tăng dần.

  Chứng minh bất biến của vòng lặp thoả mãn đủ ba tính chất khởi tạo, duy trì và kết thúc:

  - Khởi tạo: Trước khi bắt đầu vòng lặp, i = 0, nghĩa là chưa có phần tử nào được sắp xếp. Do đó, bất biến đúng với i = 0.
  - Duy trì: Giả sử bất biến đúng với i-1, tức là i-1 phần tử đầu tiên trong mảng đã được sắp xếp theo thứ tự tăng dần. Trong vòng lặp thứ i, tìm phần tử nhỏ nhất trong các phần tử từ i đến n-1 và đổi chỗ nó với phần tử a[i]. Do đó, phần tử a[i] đã được sắp xếp đúng vị trí và i phần tử đầu tiên vẫn được sắp xếp theo thứ tự tăng dần. Vì vậy, bất biến vẫn đúng với i.

- Kết thúc: Sau khi vòng lặp kết thúc, i = n-1, nghĩa là toàn bộ mảng đã được sắp xếp theo thứ tự tăng dần. Do đó, bất biến đúng với i = n-1.

Vậy, bất biến của vòng lặp trong thuật toán sắp xếp chọn được duy trì và thoả mãn đủ ba tính chất khởi tạo, duy trì và kết thúc, do đó thuật toán là đúng đắn.