TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

————-oOo————-

[HW05] PHÂN TÍCH THỰC NGHIỆM

Giảng viên hướng dẫn: ThS. Huỳnh Thị Thanh Thương

Nhóm sinh viên:

1. Phan Thanh Hải 18520705

TP. HỒ CHÍ MINH, 12/02/2020

Mục lục

Bài tập 1

Dùng kỹ thuật toán sơ cấp để đếm số phép gán và số phép so sánh trong các đoạn chương trình sau. Sau đó suy ra độ phức tạp.

Viết chương trình thực nghiệm để đếm số phép gán và số phép so sánh ở các câu trên, qua đó kiểm tra lại phần đếm lý thuyết.

a.

```
float Alpha (float x, long n)
{   long i = 1; float z = 0;
    while (i ≤ n)
    {   long j = 1; float t = 1;
        while (j ≤ i)
        {   t = t*x;
            j = 2*j;
        }
        z = z + i*t;
        i = i + 1;
    }
    return z;
}
```

Gọi α_i là số lần lặp của vòng while trong (tính độc lập với vòng while ngoài).

```
float Alpha (float x, long n)
{   long i = 1; float z = 0;            // 2 gán
    while (i ≤ n)                       // n + 1 so sánh
    {   long j = 1; float t = 1;        // 2n gán
        while (j ≤ i)                   // α_i + 1 so sánh
        {   t = t*x;                    // α_i gán
            j = 2*j;                    // α_i gán
        }
        z = z + i*t;                    // n gán
        i = i + 1;                      // n gán
    }
    return z;
}
```

Số phép gán:

$$\text{Gan}(n) = 2 + 4n + 2\sum_{i=1}^{n} \alpha_i$$

Số phép so sánh:

$$\text{SoSanh}(n) = n + 1 + \sum_{i=1}^{n} (\alpha_i + 1)$$

Tính α_i?

α_i = số lần lặp của vòng while trong

= số con j chạy từ 1 đến i, bước tăng là $2j$

$= \left|\{1; 2; 4; ...; 2^{k-1}\}\right|$

$= \left|\{2^{k-1} | k \in \mathbb{N}^*, 1 \leq 2^{k-1} \leq i\}\right|$

Ta có:

$1 \leq 2^{k-1} \leq i \Leftrightarrow \log_2 1 \leq k-1 \leq \log_2 i \Leftrightarrow 1 \leq k \leq \log_2 i + 1$

Do đó $\alpha_i = \lfloor \log_2 i + 1 \rfloor$.

Kết luận:

Số phép gán:

$$\text{Gan}(n) = 2 + 4n + 2\sum_{i=1}^{n} \lfloor \log_2 i + 1 \rfloor$$

Số phép so sánh:

$$\text{SoSanh}(n) = n + 1 + \sum_{i=1}^{n} \lfloor \log_2 i + 2 \rfloor$$

Vậy $\text{Gan}(n) + \text{SoSanh}(n) = \text{O}(n \log n)$.

b.

```
float Omega (float x, long n)
{   long i = 1; float z = 0;
    while (i ≤ n)
    {   long j = 1; float t = 1;
        while (j ≤ i*i*i)
        {   t = t*x;
            j = j + 1;
        }
        z = z + i*t;
        i = 2*i;
    }
    return z;
}
```

Gọi α_i là số lần lặp của vòng while ngoài.

Tính α_i?

α_i = số lần lặp của vòng while ngoài

= số con i chạy từ 1 đến n, bước tăng là $2i$

$= \left|\{1; 2; 4; ...; 2^{k-1}\}\right|$

$= \left|\{2^{k-1} | k \in \mathbb{N}^*, 1 \leq 2^{k-1} \leq n\}\right|$

Ta có:

$1 \le 2^{k-1} \le n \Leftrightarrow \log_2 1 \le k-1 \le \log_2 n \Leftrightarrow 1 \le k \le \log_2 n + 1$

Do đó $\alpha_i = \lfloor \log_2 n + 1 \rfloor$.

```
float Omega (float x, long n)
{   long i = 1; float z = 0;          // 2 gán
    while (i ≤ n)                     // ⌊log2 n + 2⌋ so sánh
    {   long j = 1; float t = 1;      // ⌊2 log2 n + 2⌋ gán
        while (j ≤ i*i*i)
        {   t = t*x;
            j = j + 1;
        }
        z = z + i*t;                  // ⌊log2 n + 1⌋ gán
        i = 2*i;                      // ⌊log2 n + 1⌋ gán
    }
    return z;
}
```

Xét độc lập vòng while trong với vòng while ngoài, ta có số phép gán gán và số phép so sánh lần lượt là $2i^3$ và $i^3 + 1$.

Xét vòng while trong đặt trong vòng while ngoài, vì i có bước tăng là $2i$ nên ta có:

Với vòng lặp thứ 1 thì $i = 1$.

Với vòng lặp thứ 2 thì $i = 1.2 = 2$.

Với vòng lặp thứ 3 thì $i = 2.2 = 4$...

Với vòng lặp thứ k thì $i = 2^{k-1}$.

Có tất cả $\lfloor \log_2 n + 1 \rfloor$ lần lặp nên số phép gán và số phép so sánh lần lượt của vòng while trong là $2 \sum_{i=1}^{\lfloor \log_2 n+1 \rfloor} 2^{3i-3}$ và $\sum_{i=1}^{\lfloor \log_2 n+1 \rfloor} \left(2^{3i-3} + 1\right)$.

Kết luận:

Số phép gán:

$$\begin{aligned}
\text{Gan}(n) &= 2 + 4\lfloor \log_2 n \rfloor + 4 + 2 \sum_{i=1}^{\lfloor \log_2 n+1 \rfloor} 2^{3i-3} \\
&= 4\lfloor \log_2 n \rfloor + 6 + \frac{2}{7}\left(8^{\lfloor \log_2 n+1 \rfloor} - 1\right) \\
&= \frac{2}{7}\left(8n^3 - 1\right) + 4\lfloor \log_2 n \rfloor + 6 \qquad \left(\text{với } n = 2^{\lfloor \log_2 n \rfloor}\right)
\end{aligned}$$

Số phép so sánh:

$$\begin{aligned}
\text{SoSanh}(n) &= \lfloor \log_2 n \rfloor + 2 + \sum_{i=1}^{\lfloor \log_2 n+1 \rfloor} \left(2^{3i-3} + 1\right) \\
&= \lfloor \log_2 n \rfloor + 2 + \frac{1}{7}\left(8^{\lfloor \log_2 n+1 \rfloor} - 1\right) + \lfloor \log_2 n \rfloor + 1 \\
&= \frac{1}{7}\left(8n^3 - 1\right) + 2\lfloor \log_2 n \rfloor + 3 \qquad \left(\text{với } n = 2^{\lfloor \log_2 n \rfloor}\right)
\end{aligned}$$

Vậy Gan(n) + SoSanh(n) = O(n^3).

c.

```
i = 1; ret = 0; s = 0;
while (i ≤ n)
{   j = 1;
    s = s + 1/i;
    while (j ≤ s)
    {   ret = ret + i*j;
        j = j + 1;
    }
    i = i + 1;
}
```

Gọi α_i là số lần lặp của vòng while trong (tính độc lập với vòng while ngoài).

```
i = 1; ret = 0; s = 0;          // 3 gán
while (i ≤ n)                   // n + 1 so sánh
{   j = 1;                      // n gán
    s = s + 1/i;                // n gán
    while (j ≤ s)               // αi + 1 so sánh
    {   ret = ret + i*j;        // αi gán
        j = j + 1;              // αi gán
    }
    i = i + 1;                  // n gán
}
```

Số phép gán:

$$\text{Gan}(n) = 3 + 3n + 2\sum_{i=1}^{n} \alpha_i$$

Số phép so sánh:

$$\text{SoSanh}(n) = n + 1 + \sum_{i=1}^{n} (\alpha_i + 1)$$

Tính α_i?

α_i = số lần lặp của vòng while trong

= số con j chạy từ 1 đến s, bước tăng là $j + 1$

Ta có:

$$s_i = s_{i-1} + \frac{1}{i} = s_{i-2} + \frac{1}{i-1} + \frac{1}{i} = ... = 1 + \frac{1}{2} + ... + \frac{1}{i} = \sum_{k=1}^{i} \frac{1}{k}$$

Do đó $\alpha_i = \sum_{k=1}^{i} \frac{1}{k}$.

Kết luận:

Số phép gán:

$$\text{Gan}(n) = 3 + 3n + 2\sum_{i=1}^{n}\sum_{k=1}^{i}\frac{1}{k}$$

Số phép so sánh:

$$\text{SoSanh}(n) = n + 1 + \sum_{i=1}^{n}\left(\sum_{k=1}^{i}\frac{1}{k} + 1\right)$$

Vậy $\text{Gan}(n) + \text{SoSanh}(n) = \text{O}(n \log n)$.

d.

```
i = 1;
count = 0;
while (i ≤ 3*n)
{
    x = 2*n - i;
    y = i - n;
    j = 1;
    while (j ≤ x)
    {
        if (j ≥ n)
            count = count - 1;
        j = j + 1;
    }
    if (y > 0)
        if (x > 0)
            count = count + 1;
    i = i + 1;
}
```

Gọi α_i là số lần lặp của vòng while trong (tính độc lập với vòng while ngoài).

```
i = 1;                              // 1 gán
count = 0;                          // 1 gán
while (i ≤ 3*n)                     // 3n + 1 so sánh
{
    x = 2*n - i;                    // 3n gán
    y = i - n;                      // 3n gán
    j = 1;                          // 3n gán
    while (j ≤ x)                   // αi + 1 so sánh
    {
        if (j ≥ n)                  // αi so sánh
            count = count - 1;      // (*)
        j = j + 1;                  // αi gán
    }
    if (y > 0)                      // 3n so sánh
        if (x > 0)                  // (**)
```

```
            count = count + 1;   // (***)
    i = i + 1;                   // 3n gán
}
```

Tính α_i?

α_i = số lần lặp của vòng while trong

= số con j chạy từ 1 đến x, bước tăng là $j+1$

$= x - 1 + 1 = 2n - i$

Vòng lặp while trong chỉ được thực hiện $\Leftrightarrow x \geq 1 \Leftrightarrow 2n - i \geq 1 \Leftrightarrow i \leq 2n - 1$

Số lần thực hiện lệnh (*) = số con j lớn hơn hoặc bằng n

Mà j đi từ 1 đến $2n - i$

Do đó, số lần thực hiện lệnh (*) $= (2n - i) - n + 1 = n - i + 1$, với $i \leq n$.

Xét số lần thực hiện lệnh (**) và (***). Ta có:

i	0		n		$2n$		$3n$
$y = i - n$		$-$	0	$+$		$+$	
$x = 2n - i$		$+$		$+$	0	$-$	

Số lần thực hiện lệnh (**) = số con i làm cho $y > 0$

= số con i đi từ $n+1$ đến $3n$

$= 3n - (n+1) + 1 = 2n.$

Số lần thực hiện lệnh (***) = số con i làm cho $y > 0$ và $x > 0$

= số con i đi từ $n+1$ đến $2n-1$

$= (2n-1) - (n+1) + 1 = n - 1.$

Kết luận:

Số phép gán:

$$\text{Gan}(n) = 2 + 12n + (n-1) + \sum_{i=1}^{n}(n - i + 1) + \sum_{i=1}^{2n-1}(2n - i)$$

$$= 13n + 1 + \frac{1}{2}n(n+1) + n(2n-1) = \frac{5}{2}n^2 + \frac{25}{2}n + 1$$

Số phép so sánh:

$$\text{SoSanh}(n) = 3n + 1 + 3n + 2n + \sum_{i=1}^{2n-1}[(2n - i + 1) + (2n - i)] + \sum_{i=2n}^{3n} 1$$

$$= 8n + 1 + \sum_{i=1}^{2n-1}(4n - 2i + 1) + (n+1)$$

$$= 8n + 1 + (4n^2 - 1) + (n+1) = 4n^2 + 9n + 1$$

Vậy $\text{Gan}(n) + \text{SoSanh}(n) = \text{O}(n^2)$.

Bài tập 2

Phân tích độ phức tạp của thuật toán binary search (tìm kiếm nhị phân) trong trường hợp worst case. Với thuật toán binary search mỗi input là một mảng số nguyên **có thứ tự** và một số nguyên cần tìm. Độ dài của mảng được xem là kích thước của input. Trường hợp worst case của binary search là khi số này không có trong mảng.

- Phân tích dựa trên thời gian thực thi, trường hợp worst case.
- Mẫu thử nghiệm (sample) có ít nhất 30 trường hợp (instance) với kích thước khác nhau.
- Trường hợp nhỏ nhất có kích thước 100 phần tử.
- Trường hợp sau kích thước lớn hơn trường hợp trước 50%.
- Ứng với mỗi trường hợp phát sinh 10 input với giá trị ngẫu nhiên khác nhau và tính thời gian trung bình cho 10 trường hợp này.

a. Thuật toán

BinarySearch$(A[0..n-1, x])$

```
1.  l = 0;
2.  r = n − 1;
3.  while l ≤ r do
4.      m = (l + r)/2;
5.      if x = A[m]
6.          return m;
7.      else
8.          if x < A[m]
9.              r = m − 1;
10.         else
11.             l = m + 1;
12. return −1;
```

Trường hợp worst case của binary search là khi số cần tìm không có trong mảng.

b. Cách phát sinh dữ liệu thử nghiệm

Mẫu thử nghiệm (sample) có 30 trường hợp (instance) với kích thước khác nhau.

Trường hợp nhỏ nhất có kích thước 100 phần tử.

Trường hợp sau kích thước lớn hơn trường hợp trước 50

Ứng với mỗi trường hợp phát sinh 10 input với giá trị ngẫu nhiên khác nhau, mỗi input là một mảng mà các giá trị trong mảng đi từ 0 đến 999999 và số cần tìm x (giá trị của số cần tìm x cũng đi từ 0 đến 999999).

Khi phát sinh giá trị ngẫu nhiên trong mảng ta sẽ đánh dấu giá trị đó và khi phát sinh giá trị ngẫu nhiên cho số cần tìm x, ta kiểm tra giá trị đó đã được đánh dấu

chưa. Nếu đã đánh dấu thì ta phát sinh lại giá trị ngẫu nhiên cho số cần tìm x cho đến khi giá trị của nó chưa được đánh dấu trong mảng.

c. Cách tiến hành thử nghiệm

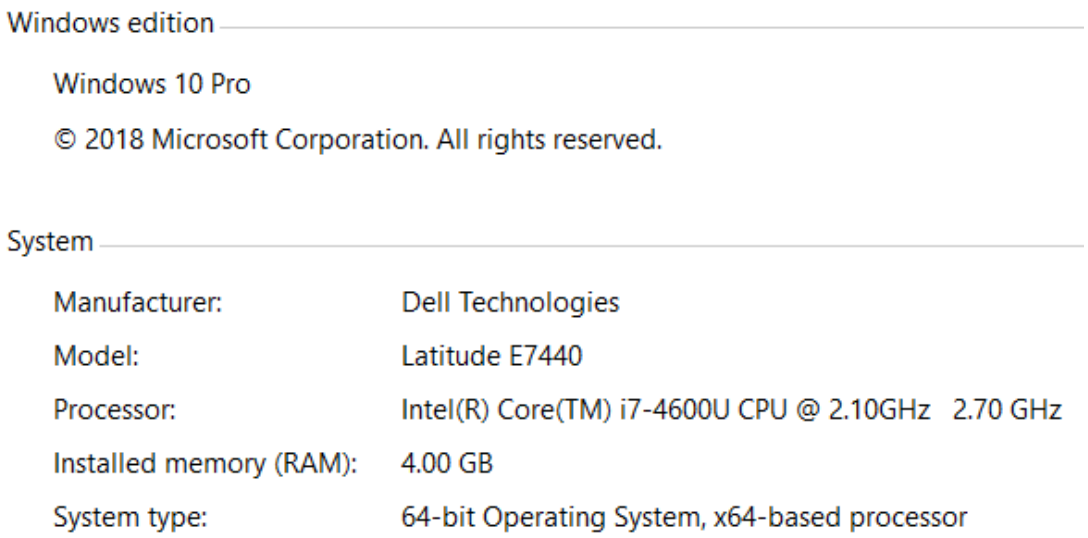
Phân tích dựa trên thời gian thực thi, trường hợp worst case.

Thời gian thực thi được tính bằng giây, xác định bằng hiệu thời gian hoàn tất gọi hàm t_{finish} và thời gian bắt đầu gọi hàm t_{start}.

Ta sẽ viết chương trình tự nhập input và xuất thời gian thực thi ra màn hình. Sau đó copy output vào file Excel.

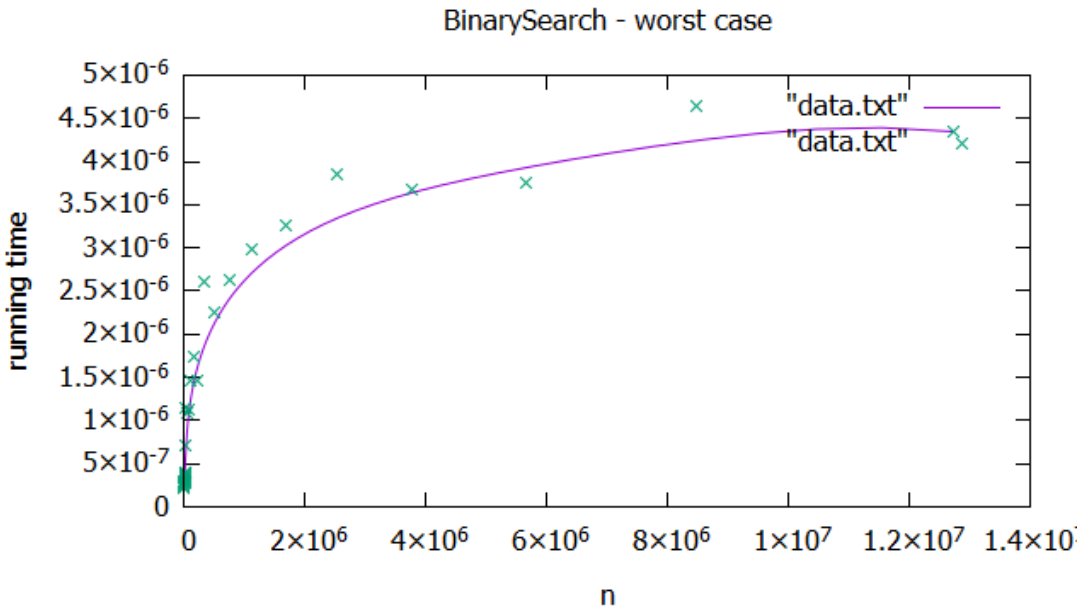
Dựa vào kết quả output trong file Excel, ta tính thời gian trung bình cho mỗi trường hợp tương ứng và xuất kết quả vào file text.

d. Cấu hình máy chạy thử nghiệm



e. Phân tích và kết luận độ phức tạp của thuật toán

Ta sẽ vẽ biểu đồ dựa vào dữ liệu thử nghiệm đã thu được.



Dựa vào biểu đồ, ta dự đoán độ phức tạp của thuật toán binary search trong trường hợp trung bình là $O(\log n)$.

Bài tập 3

Phân tích thuật toán quick sort trong trường hợp worst case.

- Phân tích dựa trên **số phép so sánh**, trường hợp worst case.
- Mẫu thử nghiệm (sample) có ít nhất 40 trường hợp (instance) với kích thước khác nhau.
- Trường hợp nhỏ nhất có kích thước 10 phần tử.
- Trường hợp sau kích thước lớn hơn trường hợp trước 30%.

a. Thuật toán

LomutoPartition$(A[0..n-1, l, r])$

1. $pivot = A[r]$;
2. $i = l - 1$;
3. **for** $j = l$ **to** $r - 1$ **do**
4. **if** $A[j] \leq pivot$
5. $i = i + 1$;
6. Hoán vị phần tử $A[i]$ với $A[j]$;
7. Hoán vị phần tử $A[i + 1]$ với $A[r]$;
8. **return** $i + 1$;

QuickSort$(A[0..n-1, l, r])$

1. **if** $l < r$
2. $p =$ **LomutoPartition**(A, p, r);
3. **QuickSort**$(A, l, p - 1)$;
4. **QuickSort**$(A, p + 1, r)$;

Trường hợp worst case của quick sort là khi quá trình phân hoạch mất cân bằng tại mỗi bước phân hoạch, nghĩa là: dãy phân hoạch một phần chỉ có 1 phần tử và phần còn lại gồm có $n - 1$ phần tử.

Thông thường, trường hợp worst case xảy ra khi input có một trong các đặc điểm sau:

1. Input đã được sắp xếp sẵn (tăng dần hoặc giảm dần).
2. Các phần tử trong input có giá trị như nhau.

b. Cách phát sinh dữ liệu thử nghiệm

Mẫu thử nghiệm (sample) có 40 trường hợp (instance) với kích thước khác nhau.

Trường hợp nhỏ nhất có kích thước 10 phần tử.

Trường hợp sau kích thước lớn hơn trường hợp trước 30

Ứng với mỗi trường hợp phát sinh một input là một mảng mà các giá trị trong mảng đi từ 0 đến 9999.

Để dữ liệu thử nghiệm khiến cho thuật toán quick sort rơi vào trường hợp worst

case thì ta sẽ tiến hành sắp xếp mảng trước khi gọi hàm **QuickSort**.

c. Cách tiến hành thử nghiệm

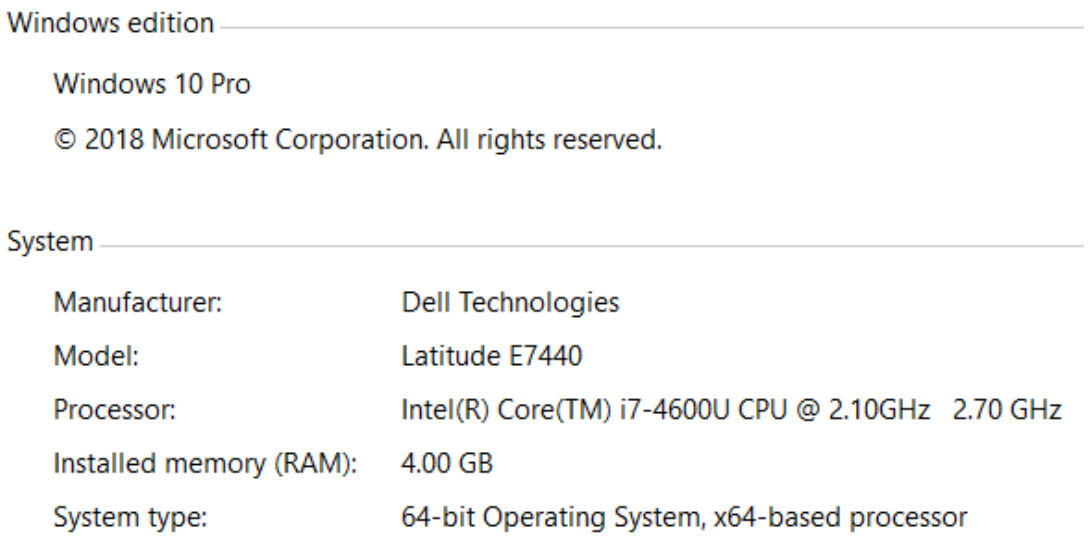
Phân tích dựa trên thời gian số phép so sánh, trường hợp worst case.

Số phép so sánh của thuật toán QuickSort được tính bằng tổng số phép so sánh được thực hiện tại dòng 4 của hàm **LomutoPartition** và dòng 1 của hàm **QuickSort**.

Ta sẽ viết chương trình tự nhập input và xuất số phép so sánh ra màn hình. Sau đó copy output vào file Excel.

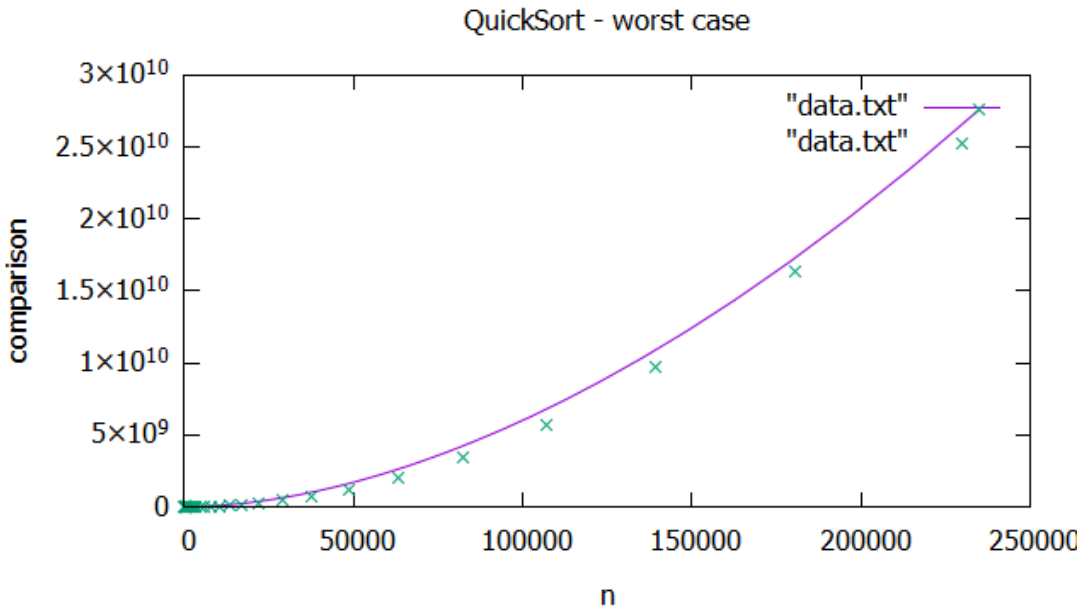
Dựa vào kết quả output trong file Excel, ta xuất kết quả vào file text.

d. Cấu hình máy chạy thử nghiệm



e. Phân tích và kết luận độ phức tạp của thuật toán

Ta sẽ vẽ biểu đồ dựa vào dữ liệu thử nghiệm đã thu được.



Dựa vào biểu đồ, ta dự đoán độ phức tạp của thuật toán quick sort trong trường hợp worst case là $O(n^2)$.

Bài tập 4

Phân tích thuật toán quick sort trong trường hợp average case.

- Phân tích dựa trên số thời gian chạy, trường hợp average case.
- Mẫu thử nghiệm (sample) có ít nhất 50 trường hợp (instance) với kích thước khác nhau.
- Trường hợp nhỏ nhất có kích thước 10 phần tử.
- Trường hợp sau kích thước lớn hơn trường hợp trước 30%.

a. Thuật toán

LomutoPartition$(A[0..n-1, l, r])$

1. $pivot = A[r]$;
2. $i = l - 1$;
3. **for** $j = l$ **to** $r - 1$ **do**
4. **if** $A[j] \leq pivot$
5. $i = i + 1$;
6. Hoán vị phần tử $A[i]$ với $A[j]$;
7. Hoán vị phần tử $A[i + 1]$ với $A[r]$;
8. **return** $i + 1$;

QuickSort$(A[0..n-1, l, r])$

1. **if** $l < r$
2. $p =$ **LomutoPartition**(A, p, r);
3. **QuickSort**$(A, l, p - 1)$;
4. **QuickSort**$(A, p + 1, r)$;

b. Cách phát sinh dữ liệu thử nghiệm

Mẫu thử nghiệm (sample) có 50 trường hợp (instance) với kích thước khác nhau.

Trường hợp nhỏ nhất có kích thước 10 phần tử.

Trường hợp sau kích thước lớn hơn trường hợp trước 30%.

Ứng với mỗi trường hợp phát sinh 10 input với giá trị ngẫu nhiên khác nhau, mỗi input là một mảng mà các giá trị trong mảng đi từ 0 đến 99999.

c. Cách tiến hành thử nghiệm

Phân tích dựa trên số thời gian chạy, trường hợp average case.

Thời gian chạy được tính bằng giây, xác định bằng hiệu thời gian hoàn tất gọi hàm t_{finish} và thời gian bắt đầu gọi hàm t_{start}.

Ta sẽ viết chương trình tự nhập input và xuất thời gian chạy ra màn hình. Sau đó copy output vào file Excel.

Dựa vào kết quả output trong file Excel, ta tính thời gian trung bình cho mỗi trường hợp tương ứng và xuất kết quả vào file text.

d. Cấu hình máy chạy thử nghiệm

Windows edition

Windows 10 Pro

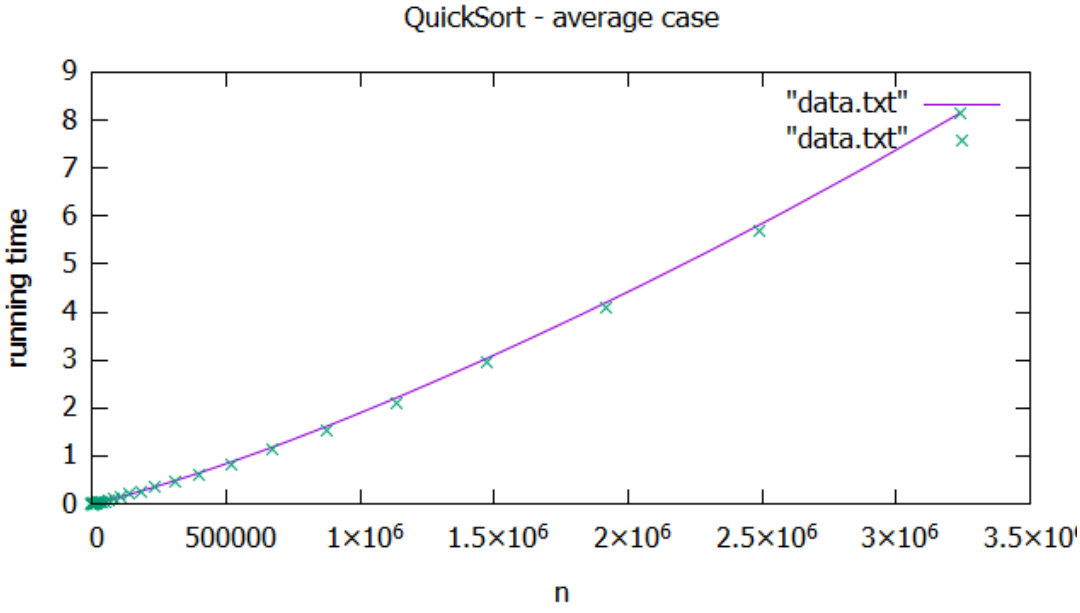
© 2018 Microsoft Corporation. All rights reserved.

System

Manufacturer:	Dell Technologies
Model:	Latitude E7440
Processor:	Intel(R) Core(TM) i7-4600U CPU @ 2.10GHz 2.70 GHz
Installed memory (RAM):	4.00 GB
System type:	64-bit Operating System, x64-based processor

e. Phân tích và kết luận độ phức tạp của thuật toán

Ta sẽ vẽ biểu đồ dựa vào dữ liệu thử nghiệm đã thu được.



Dựa vào biểu đồ, ta dự đoán độ phức tạp của thuật toán quick sort trong trường hợp average case là $O(n \log n)$.

Bài tập 5

Phân tích thuật toán shell sort trong trường hợp average case.

- Phân tích dựa trên số thời gian chạy, trường hợp average case.
- Mẫu thử nghiệm (sample) có ít nhất 50 trường hợp (instance) với kích thước khác nhau.
- Trường hợp nhỏ nhất có kích thước 10 phần tử.
- Trường hợp sau kích thước lớn hơn trường hợp trước 30%.

a. Thuật toán

ShellSort$(A[0..n-1])$

```
1.  gap = n/2;
2.  while gan > 0 do
3.      for i = gap to n − 1 do
4.          temp = A[i];
5.          j = i;
6.          while j ≥ gap and A[j − gap] > temp do
7.              A[j] = A[j − gap];
8.              j = j − gap;
9.          A[j] = temp;
10.     gap = gap/2;
```

b. Cách phát sinh dữ liệu thử nghiệm

Mẫu thử nghiệm (sample) có 50 trường hợp (instance) với kích thước khác nhau.

Trường hợp nhỏ nhất có kích thước 100 phần tử.

Trường hợp sau kích thước lớn hơn trường hợp trước 30%.

Ứng với mỗi trường hợp phát sinh 10 input với giá trị ngẫu nhiên khác nhau, mỗi input là một mảng mà các giá trị trong mảng đi từ 0 đến 99999.

c. Cách tiến hành thử nghiệm

Phân tích dựa trên số thời gian chạy, trường hợp average case.

Thời gian chạy được tính bằng giây, xác định bằng hiệu thời gian hoàn tất gọi hàm t_{finish} và thời gian bắt đầu gọi hàm t_{start}.

Ta sẽ viết chương trình tự nhập input và xuất thời gian ra màn hình. Sau đó copy output vào file Excel.

Dựa vào kết quả output trong file Excel, ta tính thời gian trung bình cho mỗi trường hợp tương ứng và xuất kết quả vào file text.

d. Cấu hình máy chạy thử nghiệm

Windows edition

Windows 10 Pro

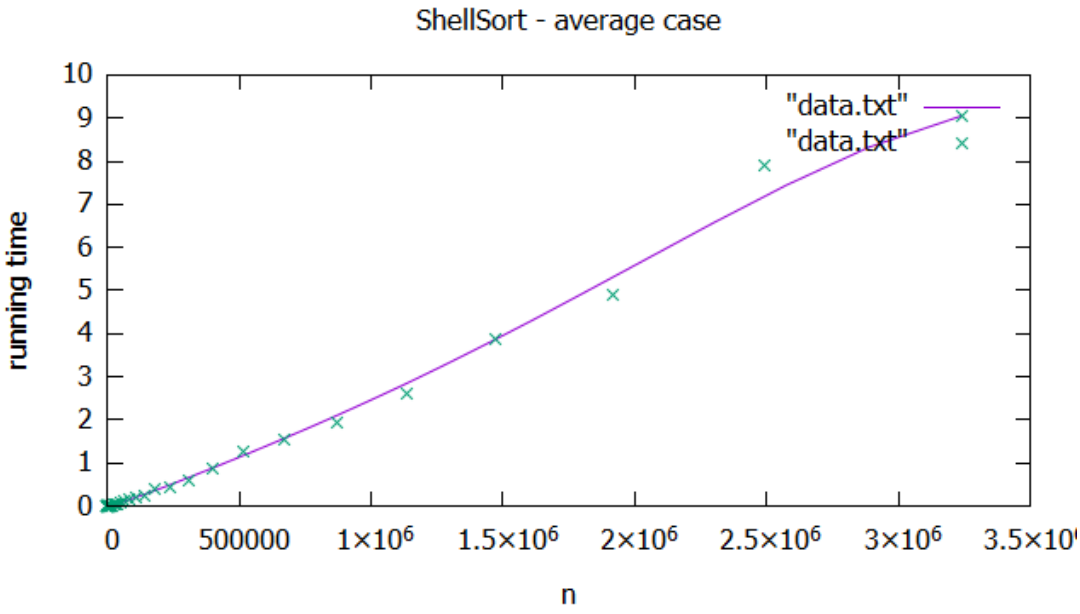
© 2018 Microsoft Corporation. All rights reserved.

System

Manufacturer:	Dell Technologies
Model:	Latitude E7440
Processor:	Intel(R) Core(TM) i7-4600U CPU @ 2.10GHz 2.70 GHz
Installed memory (RAM):	4.00 GB
System type:	64-bit Operating System, x64-based processor

e. Phân tích và kết luận độ phức tạp của thuật toán

Ta sẽ vẽ biểu đồ dựa vào dữ liệu thử nghiệm đã thu được.



Dựa vào biểu đồ, ta dự đoán độ phức tạp của thuật toán shell sort trong trường hợp average case là $O(n^2)$.

Bài tập 6

Với mỗi n nguyên dương, xem 2 thuật toán viết bằng mã giả sau đây:

Thuật toán 1.

```
i = 1; m = n - n*(n mod 2);
s = 3;
while (i ≤ m)
{   j = 1;
    k = i*i*(i mod 2);
    s = s + 3;
    while (j ≤ k)
    {   j = j + 1;
        s = s + 2;
    }
    i = i + 1;
    s = s + 2;
}
```

Thuật toán 2.

```
i = 1; m = n*(n mod 2);
s = 3;
while (i ≤ m)
{   j = 1;
    k = i*i - i*i*(i mod 2);
    s = s + 3;
    while (j ≤ k)
    {   j = j + 1;
        s = s + 2;
    }
    i = i + 1;
    s = s + 2;
}
```

Giả sử ta chỉ quan tâm đến số phép gán và số phép so sánh được thực hiện trong mỗi thuật toán. Gọi Gan1(n), SoSanh1(n), Gan2(n), SoSanh2(n) lần lượt là số phép gán và số phép so sánh của thuật toán 1 và 2.

a.

Tính theo n số phép gán trong mỗi thuật toán.

Xét thuật toán 1.

a. Nếu n là số lẻ thì $m = n - n * (n \mod 2) = n - n = 0$.

Khi đó vào điều kiện trong vòng while ngoài thì điều kiện $i \leq m$ $(1 \leq 0)$ là sai, nên không thực hiện vòng lặp while ngoài.

Do đó, với n là số lẻ thì số phép gán là Gan1(n) = 3.

b. Nếu n là số chẵn thì $m = n - n * (n \mod 2) = n - 0 = n$.

Gọi α_i là số lần lặp của vòng while trong (tính độc lập với vòng while ngoài).

```
i = 1; m = n - n*(n mod 2);          // 2 gán
s = 3;                               // 1 gán
while (i ≤ m)                        // n + 1 so sánh
{   j = 1;                           // n gán
    k = i*i*(i mod 2);               // n gán
    s = s + 3;                       // n gán
    while (j ≤ k)                    // αi + 1 so sánh
    {   j = j + 1;                   // αi gán
        s = s + 2;                   // αi gán
    }
    i = i + 1;                       // n gán
    s = s + 2;                       // n gán
}
```

Tính α_i?

α_i = số lần lặp của vòng while trong

= số con j chạy từ 1 đến k, bước tăng là $j + 1$

Ta có:

Với i là số lẻ $(i = 2a - 1, a \geq 1)$ thì $k = i * i * (i \mod 2) = i^2$.

Khi đó thì $\alpha_i = i^2 = (2a - 1)^2$.

Với i là số chẵn thì $k = i * i * (i \mod 2) = 0$. Khi đó thì $\alpha_i = 0$.

Do đó, với n là số chẵn thì số phép gán là

$$\begin{aligned}\text{Gan1}(n) &= 3 + 5n + 2\sum_{i=1}^{\frac{n}{2}}(2i-1)^2 \\ &= 3 + 5n + \frac{1}{3}n(n^2-1)\end{aligned}$$

$$\text{Vậy Gan1}(n) = \begin{cases} 3 & \text{nếu } n \text{ lẻ} \\ 3 + 5n + \frac{1}{3}n(n^2-1) & \text{nếu } n \text{ chẵn} \end{cases}$$

Xét thuật toán 2.

Thuật toán 2 chỉ có thay đổi giá trị m và k tùy theo giá trị của n và i là chẵn hay lẻ, còn lại làm tương tự như thuật toán 1.

$$\text{Vậy Gan2}(n) = \begin{cases} 3 & \text{nếu } n \text{ chẵn} \\ 3 + 5n + \frac{1}{3}n(n^2-1) & \text{nếu } n \text{ lẻ} \end{cases}$$

b.

Tính theo n số phép so sánh trong mỗi thuật toán.

Xét thuật toán 1.

a. Nếu n là số lẻ thì $m = n - n * (n \mod 2) = n - n = 0$.

Khi đó vào điều kiện trong vòng while ngoài thì điều kiện $i \leq m$ $(1 \leq 0)$ là sai, nên không thực hiện vòng lặp while ngoài.

Do đó, với n là số lẻ thì số phép gán là $\text{SoSanh1}(n) = 1$.

b. Nếu n là số chẵn thì $m = n - n * (n \mod 2) = n - 0 = n$.

Sử dụng kết quả đã có ở câu a), ta được:

Với n là số chẵn thì số phép so sánh là

$$\begin{aligned}\text{SoSanh1}(n) &= n + 1 + \sum_{i=1}^{\lceil \frac{n}{2} \rceil} \left[(2i-1)^2 + 1\right] + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} 1 \\ &= n + 1 + \left\lceil \frac{1}{6} n(n^2+2) \right\rceil + \frac{1}{2} n \\ &= 1 + \frac{3}{2} n + \left\lceil \frac{1}{6} n(n^2+2) \right\rceil\end{aligned}$$

$$\text{Vậy SoSanh1}(n) = \begin{cases} 1 & \text{nếu } n \text{ lẻ} \\ 1 + \frac{3}{2} n + \left\lceil \frac{1}{6} n(n^2+2) \right\rceil & \text{nếu } n \text{ chẵn} \end{cases}$$

Xét thuật toán 2.

Thuật toán 2 chỉ có thay đổi giá trị m và k tùy theo giá trị của n và i là chẵn hay lẻ, còn lại làm tương tự như thuật toán 1.

$$\text{Vậy SoSanh2}(n) = \begin{cases} 1 & \text{nếu } n \text{ chẵn} \\ 1 + \frac{3}{2} n + \left\lceil \frac{1}{6} n(n^2+2) \right\rceil & \text{nếu } n \text{ lẻ} \end{cases}$$

c.

Suy ra độ phức tạp của mỗi thuật toán.

Xét thuật toán 1.

$\text{Gan1}(n) + \text{SoSanh1}(n) = \text{O}(n^3)$ (xét chung cho cả trường hợp n chẵn và lẻ)

Xét thuật toán 2.

$\text{Gan2}(n) + \text{SoSanh2}(n) = \text{O}(n^3)$ (xét chung cho cả trường hợp n chẵn và lẻ)

d.

Có thể so sánh độ phức tạp của 2 thuật toán trên hay không?

e.

Cho biết ý nghĩa của biến s trong mỗi thuật toán.

Ý nghĩa của biến s là dùng để đếm số phép gán (bao gồm phép gán lên biến s) của mỗi thuật toán.

f.

Không dùng các công thức đã tính ở câu a, viết chương trình để tính trực tiếp giá trị của các hàm Gan1(n), Gan2(n) với $n = 1, 2, ..., 20$. Sau đó in ra dạng bảng để so sánh kết quả với công thức đã có nhờ câu a.

Giá trị n	Gan1(n) (tính bằng công thức)	Gan2(n) (tính bằng công thức)	Số phép gán thực tế của thuật toán 1	Số phép gán thực tế của thuật toán 2
1	3	8	3	8
2	15	3	15	3
3	3	26	3	26
4	43	3	43	3
5	3	68	3	68
6	103	3	103	3
7	3	150	3	150
8	211	3	211	3
9	3	288	3	288
10	383	3	383	3
11	3	498	3	498
12	635	3	635	3
13	3	796	3	796
14	983	3	983	3
15	3	1198	3	1198
16	1443	3	1443	3
17	3	1720	3	1720
18	2031	3	2031	3
19	3	2378	3	2378
20	2763	3	2763	3

g.

Với mỗi n, gọi p(n) là số nguyên tố thứ n (nghĩa là p(1) = 2, p(2) = 3, p(3) = 5...). Gọi K(n) là thuật toán có được từ thuật toán 1 bằng cách thay giá trị của n bởi p(n). Hãy xác định số phép gán và số phép so sánh được thực hiện bởi thuật toán K và suy ra độ phức tạp của thuật toán này.

Với $n \geq 2$ thì p(n) luôn là số lẻ.
Khi đó, dựa vào kết quả của câu a. và câu b. thì số phép gán và số phép so sánh lần lượt là 3 và 1 (không phụ thuộc vào n).
Vậy độ phức tạp của thuật toán K là O(1).

h.

Gọi H(n) là thuật toán có được từ thuật toán 1 bằng cách thay đổi giá trị của n bởi $n(n+1)$. Hãy xác định số phép gán và số phép so sánh được thực hiện bởi thuật toán H và suy ra độ phức tạp của H.

Nếu thay n bởi $n(n+1)$ thì ta thấy $n(n+1)$ luôn là số chẵn.
Khi đó, dựa vào kết quả của câu a. và b. thì số phép gán và số phép so sánh lần lượt là $3+5n(n+1)+\frac{1}{3}n(n+1)[n^2(n+1)^2-1]$ và $1+\frac{3}{2}n(n+1)+\frac{1}{6}n(n+1)[n^2(n+1)^2+2]$.
Vậy độ phức tạp của thuật toán H là O(n^6).