Nguyễn Thanh Đạt - B19DCVT081

Câu 1:

- Giải thuật hay thuật toán là dãy các thao tác sơ cấp trên tập dữ liệu đầu vào (Input) để đưa ra kết quả (output)

+ Thuật toán phải đảm bảo các tính chất sau:

* Tính đơn định.
* Tính đúng
* Tính đúng đắn
* Tính phổ dụng.
* Tính khả thi

+ Giải thuật là khái niệm mở rộng của thuật toán với tính xác định được làm mềm đi.

- Cấu trúc dữ liệu hiểu là phương pháp biểu diễn đối tượng ở thế giới thực trong hệ thống máy tính

- Các cấu trúc dữ liệu trên định của C++

+ int: kiểu số nguyên
+ float: kiểu số thực.
+ char: kiểu ký tự.
+ array: kiểu mảng.
+ String = kiểu xâu.

- Mối quan hệ giữa cấu trúc dữ liệu và giải thuật

Cấu trúc dữ liệu + giải thuật = chương trình

Câu 2:

a. Trình bày ý tưởng thuật toán Merge sort trên mảng. So sánh thuật toán Merge sort với ít nhất 2 thuật toán khác.

* Các thuật toán sắp xếp đơn giản như Bubble sort, Insertion.... đều không thể xử lý được dữ liệu lớn. Thuật toán sắp xếp trộn lấy ý tưởng từ việc chia để trị để chia nhỏ bài toán thành các bài toán nhỏ hơn, sau đó giải quyết chúng. Từ đó sẽ giúp xử lý dữ liệu lớn một cách tốt hơn, tối ưu về mặt thời gian

* Ý tưởng đưa ra:

+ Chia danh sách gồm n phần tử chưa được sắp xếp thành n danh sách con, mỗi danh sách chứa một phần tử (danh sách mỗi phần tử được coi là sắp xếp

+ Liên tục hợp nhất các danh sách con để tạo danh sách con được sắp xếp mới cho đến khi chỉ còn lại một danh sách. Đây sẽ là danh sách đc sắp xếp

* Khi triển khai code, ta sẽ cụ thể hoá các bước:

+ Bước 1:

* Chia dãy cần sắp xếp thành 2 dãy con.
* Từ dãy con thu được lại tiếp tục chia thành 2 dãy con nhỏ hơn nữa.
* Quá trình phân chia tiếp tục cho đến khi thu được dãy con chỉ còn duy nhất 1 phần tử

* Bước 2:
+ Hoà nhập 2 dãy con nhỏ nhất thành dãy con lớn hơn sao cho đúng thứ tự
+ Từ hai dãy con lớn hơn lại hoà nhập thành 2 dãy con lớn hơn nữa ....
+ Quá trình hoà nhập cứ tiếp tục như vậy cho đến khi thu được dãy số ban đầu đã sắp xếp

mergeSort(arr[], l, r)

If r > l:

1. Tìm chỉ số nằm giữa mảng để chia mảng 2 nửa:
   middle = (l+r)/2
2. Gọi đệ quy hàm mergeSort cho nửa đầu tiên:
   mergeSort(arr, l, m)
3. Gọi đệ quy hàm mergeSort cho nửa thứ hai:
   mergeSort(arr, m+1, r)
4. Gộp 2 nửa mảng đã sắp xếp ở (2) và (3);
   merge(arr, l, m, r) được sắp xếp

Mergesort:

```
int i, j, k, n1 = m-l+1; n2 = r-m;
int L[n1], R[n2]; // tạo lập hai mảng phụ L và R.
for(i=0; i<n1; i++) // Nửa đoạn được sắp xếp thứ nhất vào L
        L[i] = Arr[l+i];
for(j=0; j<n2; j++) // Nửa đoạn được sắp thứ nhất vào R.
        R[j] = Arr[m+1+j];
// hợp nhất các mảng phụ và trả lại vào Arr[]
i=0; j=0; k=l;
while(i<n1 && j<n2){
    if(L[i]<=R[j]){Arr[k]=L[i]; i++;}
    else{Arr[k]=R[j]; j++;}
    k++;
}
while(i<n1){
    Arr[k]=L[i]; i++; k++;
}
while(j<n2){
    arr[k]=R[j]; j++; k++;
}
```

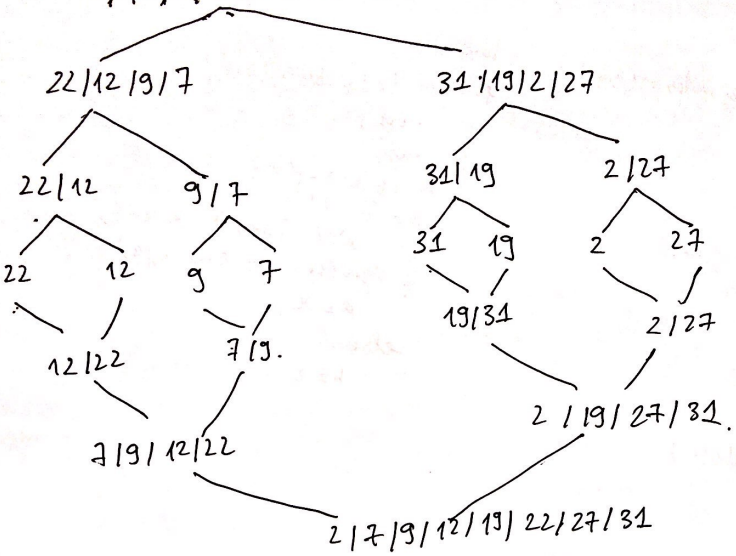
Thuật toán Merge với 2 thuật toán ≠

Thuật toán	Tốt nhất	Trung bình	Xấu nhất	Bộ nhớ
Buble sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Insection Sort	O(n)	$O(n^2)$	$o(n^2)$	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)

	Merge Sort	Insection Sort	Bubble Sort
Độ hiệu quả	- So sánh các phần tử của 2 phần tử đã sắp xếp để hợp nhất chúng thành mảng được sắp xếp cuối cùng.	- So sánh các phần tử để quyết định vị trí của một phần tử trong mảng đã được sắp xếp một phần.	- So sánh các phần tử để đặt các phần tử lớn nhất vào vị trí cuối cùng.
Độ phức tạp về thời gian	O(nlogn)	$O(n^2)$	$O(n^2)$
Thao tác so sánh	phân bổ thêm bộ nhớ và sao chép dữ liệu.	hoán đổi	hoán đổi.
Ưu điểm	- Chạy nhanh, độ phức tạp, O(N+log·N) O(Nlogn) * ổn định	* Nếu danh sách đã gần đúng thứ tự, Insection sort sẽ chạy rất nhanh.	* Code đơn giản, dễ hiểu * không tốn thêm bộ nhớ
Nhược điểm	+ cần dùng thêm bộ nhớ để lưu mảng A.	Độ phức tạp O(N2) O(N2) không đủ nhanh với dữ liệu lớn	Độ phức tạp O(N2) O(N2) không đủ nhanh với dữ liệu lớn

* Với mảng đã được sắp xếp, thì bubble sort cho tốc độ nhanh nhất do chỉ phải đọc biết trước đây là mảng có thứ tự của 2 thuật toán là O(n)
* Với mảng gần như đã được sắp xếp thì Insertion Sort và Binary Insertion Sort là những sự lựa chọn tốt nhất do số phép hoán đổi phải thực hiện ít.
* Merge sort có tốc độ ổn định xuyên suốt cả 4 loại dữ liệu đầu vào

b) 22 | 12 | 9 | 7 | 31 | 19 | 2 | 27



Câu 3:

$P(x) = x_m \cdot x^m + a_{m-1} x^{m-1} + \dots + a_1 x + a_0$.

$Q(x) = b_n x^n + b_{n-1} x^{n-1} + \dots + b_1 x + b_0$

```
if (m<n) p=m; else p=n;
for (i=0; i<=p; i++)
        c[i] = a[i] + b[i]; // thực hiện phần (p = min(n,m))
  if (p<m)
     for (i=p+1; i<=m; i++) c[i]=a[i];
  else
     for (i=p+1; i<=n; i++) c[i]=b[i];
  while (p>0 && c[p]=0) p=p-1;
```

// thực hiện $\max_{n,m} - p$ lần

→ Tổng số lần tính c[i] là $p + (\max(n,m) - p)$ lần.

→ độ phức tạp của phép tính toán c[i] là:

→ $T1 = O(\max(n,m))$

Phép toán while (p>0 && c[p]=0) trải qua phần duyệt hay $\min(n,m)$ lần duyệt → $O2 = O(\min(n,m))$.

Áp dụng quy tắc cộng.

→ $T = T_1 + T_2 = O(\max(n,m) + \min(n,m))$

Áp dụng quy tắc lấy max.

→ $T = O(\max(n,m) + \min(n,m))$

$= O(\max(\max(n,m), \min(n,m))$

$= O(\max(n,m))$ - $(\max(n,m), \min(n,m)$: chứa hằn hoặc m