Hướng dẫn giải

Câu 1

a. Đúng. Với N nhỏ, MergeSort chia đôi không có đáng kể lợi ích (vd: khi N=2), nhưng mất nhiều thời gian cho sao chép dữ liệu ra mảng phụ và gọi hàm đệ qui.

b. Sai. Linked-list không truy nhập trực tiếp được, nên thời gian là O(N).

c. Sai. Selection sort và Heap sort không phải là sắp xếp ổn định.

d. Đúng. Linked-list ngoài lưu giá trị như mảng, còn phải lưu thêm con trỏ (là địa chỉ ô nhớ).

e. Sai. Trong cây min-heap, đọc giá trị nhỏ nhất là O(1) do ở gốc cây.

Câu 2

Cách 1 (sắp xếp): O(N.log(N))

- Sắp theo thứ tự tăng dần (MergeSort, QuickSort. C++, Java: hàm có sẵn)
- Lấy phần tử đầu tiên
- Duyệt các phần tử tiếp sau, và lấy nếu nó khác với phần tử ngay trước nó

Cách 2 (Cây tìm kiếm): O(N.log(N))

- S là tập kết quả, lưu bằng cây tìm kiếm nhị phân cân bằng (C++ map, Java TreeSet).
- Duyệt lần lượt các phần tử. Phần tử được cho vào S nếu trước đó không nằm trong S.

Cách 3 (Bảng băm): Tương tự cách 2, O(N)

- S là tập kết quả, lưu bằng bảng băm (hash table) (C++ unordered_map, Java HashSet).
- Duyệt lần lượt các phần tử. Phần tử được cho vào S nếu trước đó không nằm trong S.

Chú ý: Cách làm sử dụng mảng đếm các phần tử (tương tự counting sort) không đúng, bởi cách làm đó chỉ áp dụng được khi chênh lệch giữa phần tử lớn nhất và nhỏ nhất của dãy không lớn. (Còn ở bài này không có giới hạn nào.)

Câu 3

- Mỗi nút có: count (số nút), left (child), right (child)
- Duyệt cây theo trật tự sau (post order) để tìm số nút.

```
int countNodes(Node root) {
```

```
    if (root == null) return 0;
    root.count = 1 + countNodes(root.left) + countNodes(root.right);
    return root.count;
}
```

Câu 4.

```
long long power(int k, int n) {
    if (n<0) return -1; //Hoặc gửi ra thông báo lỗi
    if (n==0) return 1;
    long long H = power(k, n/2);
    if (n%2 == 0) return H*H;
    else return k*H*H;
}
```

Chú ý: nếu không dùng biến phụ H thì chạy chậm (=>gần như sai), bởi phải tính lại nhiều lần.

~~if (n%2 == 0) return power(k, n/2)*power(k, n/2);~~

~~else return k*power(k, n/2)*power(k, n/2);~~

Câu 5.

a. Trình bày thuật toán Prim xuất phát từ 1 đỉnh (không phải thuật toán Kruskal). Tổng trọng số cây bao trùm nhỏ nhất là 17.

b. Khi tất trọng số bằng nhau. (Hay các trường hợp khác.)

Câu 6.

a. Trình bày kết quả với phương pháp chia (seperate chaining).

Trình bày kết quả với phương pháp thăm dò tuyến tính (linear probing)

b. Với mảng kích cỡ 23, không có va chạm => nên sử dụng thăm dò tuyến tính.

(Về lý thuyết, phương pháp chia dùng mảng nhỏ hơn, nên bằng 1/5 số phần tử. Ngược lại với thăm dò tuyến tính, dùng mảng lớn hơn, nên gấp 2 số phần từ.)

Câu 7.

a. Vẽ đúng

b. Thứ tự các đỉnh (và khoảng cách nhỏ nhất đi từ 0) tìm được theo Dijkstra là: 0 (0), 2 (2), 3(3), 1(4), 4(5), 7(6), 5(8), 6(14)

Câu 8.

Phần chương trình chung

Result = null //Tập lưu kết quả min

S= null//Lớp lưu tập con M phần tử, tuỳ cách cài đặt quyết định độ phức tập

for i = 1 to M-1:

    S.add(a_i, i) //Khởi taọ S gồm M-1 phần tử đầu dãy

for i= M to N:

    S.add(a_i, i) //Thêm phần tử phía đầu bên phải

    Result.add(S.getMin()); //Lấy ra giá trị nhỏ nhất của tập M phần tử

    S.remove(a_i-M, i-M); //Xoá đi phần tử bên trái

return Result

Cách 1:

- Cài đặt S là cây tìm kiếm nhị phân cân bằng (C++ là set, Java là TreeSet), mỗi node gồm hai giá trị (a_i, i)
- add(a_i, i) là thêm vào cây node (a_i, i)
- getMin(): lấy giá trị nhỏ nhất (trái nhất của cây)
- remove(a_i-M, i-M): xoá node(a_i-M, i-M)

Độ phức tạp: O(N*log(M))

Chú ý: Cách làm này (hay dùng hàng ưu tiên chỉ số) chính là kỹ thuật được dùng để cài đặt thuật toán Dijkstra (và đơn giản hơn cài đặt thuật toán Dijstra).

Cách 2:

- Cài đặt S gần như stack, nhưng cho phép loại 2 đầu, có thể cài bằng double linked-list hay mảng với 2 con trỏ đầu và cuối. (C++ và Java có deque.). Mỗi phần tử (cũng như cách 1) là một node gồm hai giá trị (a_i, i).
- add(a_i, i):
    + xoá từ phải qua trái tất cả node có giá trị lớn hơn hay bằng a_i
    + thêm vào (a_i, i)

    (Bằng cách này, S luôn có giá trị tăng dần, với phần tử lớn nhất vừa được cho vào).
- getMin(): giá trị nhỏ nhất là phần từ đầu tiên bên trái (do dãy tăng)
- remove(i-M): Nếu node đầu bên trái có chỉ số i-M thì xoá, nếu không bỏ qua.

Độ phức tạp: O(N)

Chú ý: Cách 2 có thể chỉ lưu trong S chỉ số (index) của phần tử trong mảng đầu vào, bởi từ chỉ số thì có được giá trị.