ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA: CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI THI GIỮA KỲ
THỰC HÀNH CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

Giảng viên hướng dẫn: Nguyễn Khánh Toàn
Họ và tên sinh viên: Nguyễn Hải Đăng
Mã số sinh viên: 20120049
Lớp: 20CTT1A
Niên khoá: 2021-2022

BÀI 1: (3 điểm)

Bài này có 2 hàm chính:

- Hàm `addElementAt(List& L, int index, node* pNew)`: Hàm có nhiệm vụ định vị node lại cuối (L.tail) sau đó thêm gắn node pNew vào sau vị trí node thứ $index$ của danh sách liên kết L (vị trí các node bắt đầu tính từ 1).
- Hàm `reList(List& L, int n)` : Trong hàm có vòng lặp, biến đếm i ($1 \leq i < n$), bước nhảy 2 đơn vị.

Mã giả:

```
reList(List& L, int n) {
    for (int i = 1; i < n; i += 2) {
        node* pNew = L.tail;
        addElementAt(L, i, pNew);
    }
}
```

Ý tưởng: sẽ có 1 vòng lặp trong hàm `reList(List& L, int n)` chạy từ $i = 1$ tới khi $i < n$ thì dừng lại và sau mỗi lần lặp thì i tăng lên 2 đơn vị. Trong mỗi lần lặp, node* pNew = L.tail (node cuối hiện tại của danh sách L). Rồi sau đó sẽ gọi hàm `addElementAt(L, i, pNew)`, hàm này sẽ tìm node* p là node có vị trí thứ i tromg danh sách L, sau đó hàm sẽ định vị lại node cuối sẽ là node liền trước node cuối hiện tại. Sau đó chèn node* pNew vào sau node* p đã xác định trước đó.

Độ phức tạp:

- Vòng lặp được chạy $\frac{n}{2}$ lần.
- Trong mỗi vòng lặp, độ phức tạp của hàm `addElementAt(L, i, pNew)` là $O(n)$.
- Độ phức tạp tổng quát của bài là: $O(n^2)$.

BÀI 2: (4 điểm)

Các hàm chính trong bài;

- Hàm `heapSort(vector<int>& a, vector<int>& b, vector<int>& c, int n)`: Sắp xếp vector c theo thứ tự từ bé đến lớn bằng thuật toán sắp xếp Heapsort, vector a và b sắp xếp theo cách sắp xếp vector c.
- Hàm `binarySearch(vector<int>& a, int tmp, int k, int l, int n)`: Hàm tìm kiếm nhị phân trên vector a đã sắp xếp, tìm kiếm vị trí min ($i \leq min \leq n - 1$) trên vector a thoả $k \geq a[min] + tmp$ sao cho min có giá trị nhỏ nhất. Hàm trả về giá trị min.
- Hàm `createMinus(vector<int> a, vector<int> b)` : tạo một vector c với các phần tử $c[i] = a[i] - b[i]$ ($0 \leq i < a.size()$). Hàm trả về vector c.

- Hàm `handle(vector<int>& a, vector<int>& b)` : hàm đếm số lượng cặp số (i, j) sao cho $a_i + a_j > b_i + b_j$.

Mã giả:

```
handle(vector<int>& a, vector<int>& b) {
    vector<int> c = createMinus(a, b);
    unsigned long long t = 0;
    int n = a.size();
    heapSort(a, b, c, n);
    for (int i = 0; i < n - 1; i++) {
        int b = binarySearch(c, c[i], 0, i + 1, n);
        if (b > 0) t += (n - b);
    }
    return t;
}
```

Ý tưởng:

- $a_i + a_j > b_i + b_j \Leftrightarrow (a_i - b_i) + (a_j - b_j) > 0$
- Gọi hàm `createMinus(a, b)` để tạo vector c có các phần tử như đã đề cập ở phần các hàm chính. Biến đếm số cặp (i, j) là $t = 0$, n là số phần tử của vector a.
- Sau đó, gọi hàm `heapSort(a, b, c, n)` để sắp xếp vector c tăng dần, vector a và b được sắp xếp theo cách sắp xếp của vector c.
- Trong vòng lặp, biến đếm i $(0 \leq i < n - 1)$, bước nhảy 1 đơn vị.
    - Đặt b có giá trị bé nhất $(i \leq b < n)$ thoả $c[i] + c[b] > 0$. b có thể trả về giá trị -1 nếu không có phần tử nào thoả.
    - Nếu $b > 0$ thì giá trị của biến t sẽ tăng $(n - b)$ đơn vị.
- Trả về giá trị t.

Độ phức tạp:

- Hàm `heapSort()` có độ phức tạp là $O(nlogn)$.
- Vòng lặp chạy $n - 1$ lần. Trong mỗi lần lặp, hàm `binarySearch()` có độ phức tạp là $O(logn)$.
- Vậy độ phức tạp tổng quát là: $O(nlogn)$.

<u>BÀI 3:</u> (3 điểm)

Các hàm chính trong bài;

- Hàm `heapSort(vector<int>& a, vector<int>& b, vector<int>& c, int n)`: Sắp xếp vector c theo thứ tự từ bé đến lớn bằng thuật toán sắp xếp Heapsort, vector a và b sắp xếp theo cách sắp xếp vector c.

- Hàm `binarySearch(vector<int> a, int tmp, int k, int l, int n, int t)`: Hàm đếm số lượng phần tử trên vector a (chỉ số từ 1 tới n – 1) thoả $k = a[mid] - temp$. Hàm trả về giá trị t là số lượng phần tử.
- Hàm `creTempVector(vector<int> a, int n, vector<int>& b, vector<int>& c)`: Hàm tạo ra 2 vector b và c có độ dài n với các phần tử thoả:
  - Vector b: $b[i] = a[i] + \left\lfloor (i+1)^{\frac{3}{2}} \right\rfloor$.
  - Vector c: $c[i] = a[i] - \lceil \sqrt{i+1} \rceil$.
  - $0 \leq i < n$.
- Hàm `countPairs(vector<int>& b, vector<int>& c, int n, int k)`: Hàm đếm số cặp (i, j) thoả yêu cầu đề bài. Trong hàm:
  - Heapsort vector b.
  - Heapsort vector c.
  - Vòng lặp, biến đếm i chạy trong khoảng $0 \leq i < n - 1$ với bước nhảy 1 đơn vị: đếm số cặp số (i, j) thoả yêu cầu đề bài. Mỗi cặp thoả mãn thì biến $count$ tăng lên 1 đơn vị.
  - Trả về giá trị $count$.

Mã giả:

```
countPairs(vector<int>& b, vector<int>& c, int n, int k)
{
    unsigned long long count = 0;
    int i;
    heapSort(b, n);
    heapSort(c, n);
    int t = 0;
    for (i = 0; i < n - 1; i++)
        t = binarySearch(c, b[i], k, 0, n, t);
    return t;
}
```

Ý tưởng:

- $A_j - A_i = \lfloor i^{1.5} \rfloor + \lceil \sqrt{j} \rceil \Leftrightarrow A_i + \lfloor i^{1.5} \rfloor = A_j - \lceil \sqrt{j} \rceil$
- Trong hàm main(), vector a đã được tạo:
  - Gọi hàm `creTempVector(a, n, b, c)` đã tạo vector b và c thoả yêu cầu của hàm đã được nêu ở phần các hàm chính.
  - Gọi hàm `countPairs(b, c, n, 0)` để đếm số lượng cặp số (i, j) thoả mãn yêu cầu, trong đó hàm `binarySearch(c, b[i], k, i, n)` trả về số cặp phần tử (i, j) thoả mãn đề bài (hiện tại).

Độ phức tạp:

- Heapsort có độ phức tạp là $O(nlogn)$.

- Trong vòng lặp, biến đếm i chạy từ $(0 \leq i < n - 1)$ với bước nhảy 1 đơn vị, mỗi lần lặp sẽ gọi hàm `binarySearch()` => độ phức tạp là $O(n(logn)^2)$.
- Độ phức tạp tổng quát là $O(n(logn)^2)$.

Các nguồn tham khảo

- https://www.geeksforgeeks.org/number-of-indices-pair-such-that-element-pair-sum-from-first-array-is-greater-than-second-array/