|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC SÀI GÒN  **KHOA CÔNG NGHỆ THÔNG TIN**  Bài tập Cấu trúc dữ liệu và Giải thuật  **CÁC THUẬT TOÁN SẮP XẾP**     |  |  | | --- | --- | | Danh sách sinh viên nhóm: |  | | Đặng Thành Đại | 3123411061 | | Hồ Hữu Anh Tuấn | 3123411323 | | Nguyễn Thành Vinh | 3123411342 | | Nguyễn Anh Khoa |  |   Giảng viên phụ trách: Đỗ Như Tài |

Thành Phố Hồ Chí Minh, tháng 03 năm 2025

MỤC LỤC

[I/ Phân công 4](#_Toc192786745)

[II/ Nội dung 5](#_Toc192786746)

[A. Câu hỏi 5](#_Toc192786747)

[1. Trình bày tư tưởng của các thuật toán sắp xếp? 5](#_Toc192786748)

[2. Thuật toán thích nhất ?Thuật toán không thích nhất ? 6](#_Toc192786749)

[3. Trình bày và cài đặt các thuật toán sắp xếp 6](#_Toc192786750)

[4. Ưu điểm và nhược điểm của các loại sort trên ? Cách khác phục những nhược điểm này là gì ? 12](#_Toc192786751)

[B. Bài tập cơ sở 16](#_Toc192786752)

[Bài 1: 16](#_Toc192786753)

[Bài 2: 27](#_Toc192786754)

[Bài 3: 40](#_Toc192786755)

[Bài 4: 55](#_Toc192786756)

[B. Bài tập vận dụng 62](#_Toc192786757)

[Bài 1: 62](#_Toc192786758)

[Bài 2: 65](#_Toc192786759)

[Bài 3: 67](#_Toc192786760)

[Bài 4: 71](#_Toc192786761)

[Bài 5: 74](#_Toc192786762)

[Bài 6: 76](#_Toc192786763)

[Bài 7: 78](#_Toc192786764)

[Bài 9: 79](#_Toc192786765)

# I/ Phân công

|  |  |  |
| --- | --- | --- |
| **MSSV** | **Tên** | **Nội dung phân công** |
| 3123411061 | Đặng Thành Đại | Word, sửa code  Câu hỏi: 4  BT cơ sở: 1, 2, 4  BT ứng dụng: 9 |
| 3123411323 | Hồ Hữu Anh Tuấn | Câu hỏi: 2  BT cơ sở: 3  BT ứng dụng: 1,2,3 |
| 3123411342 | Nguyễn Thành Vinh | Câu hỏi: 3  BT cơ sở: 1,2  BT ứng dụng: 4,5 |
|  |  | Câu hỏi: 1  BT cơ sở: 1, 3  BT ứng dụng: 6,7 |

# II/ Nội dung

## A. Câu hỏi

### 1. Trình bày tư tưởng của các thuật toán sắp xếp?

- Mục đích của các thuật toán sắp xếp chủ yếu là tổ chức lại các phần tử trong một dãy sao cho chúng theo một thứ tự nhất định (tăng dần hoặc giảm dần). Ý niệm của các thuật toán này có thể chia thành các cách tiếp cận sau:

1. **Sắp xếp theo so sánh giá trị**

**a/ Nghịch thế**

- Interchange Sort: tìm các cặp nghịch thế, đổi chỗ các cặp nghịch thế từ đó làm giảm đi số cặp nghịch thế.

- Bubble Sort: Từ các hiện tượng tự nhiên trong thực tế như khi thả gạo và trấu vào cốc nước, bọt khí từ nước nổi lên trên 🡪 những vật có trọng lượng riêng nhỏ hơn sẽ nổi lên trên rồi đến những vật có trọng lượng riêng lớn hơn.

- Shaker Sort: lợi dụng khuyết điểm của Bubble Sort đó là dữ liệu đã được sắp xếp thì chương trình vẫn phải duyệt qua tuần tự 🡪 thay vì phải so sánh các phần tử đã hoán đổi (đã sắp xếp) ta sẽ đánh dấu vị trí lần hoán đổi cuối cùng để lần duyệt sau không cần phải duyệt qua lần nữa.

**b/ Chèn dữ liệu**

- Insertion Sort: được mô phỏng theo cách xếp của người chơi bài tây, đó là khi cần “chèn” một lá bài vào vị trí thích hợp trong số các lá bài đã được sắp xếp trước đó.

- Binary Insertion Sort: ở Insertion Sort ta tìm kiếm vị trí thích hợp bằng tìm kiếm tuần tự, thay vào đó ta sử dụng tìm kiếm nhị phân để tìm kiếm.

- Shell Sort: một hướng tiếp cận khác của Insertion Sort bằng cách phân hoạch dãy thành nhiều dãy con đan xen nhau, sau đó sắp xếp các dãy con trước khi thực hiện chèn trực tiếp.

**c/ Chọn dữ liệu**

- Selection Sort: tìm phần tử nhỏ nhất/ lớn nhất mỗi lần duyệt để lên đầu danh sách.

- Heap Sort: ở Selection Sort, ở mỗi lần duyệt ta tìm thông tin về phần tủ nhỏ nhất/ lớn nhất những lại không tận dụng được các thông tin đó ở lần duyệt sau 🡪 ta làm sao để lưu trữ các thông tin tìm kiếm đó để có thể sử dụng lại ở các lần duyệt sau.

**d/ Chia để trị**

- Quick Sort: chọn một phần tử làm "chốt" rồi phân chia dãy thành các phần tử nhỏ hơn và lớn hơn phần tử "chốt", tiếp tục đệ quy sắp xếp các phần con.

- Merge Sort: chia dãy thành các dãy con, sắp xếp chúng rồi kết hợp lại để thu được dãy đã sắp xếp.

1. **Sắp xếp dựa trên sàng lọc**

- Radix Sort: sắp xếp các phần tử theo từng chữ số (hoặc bit) từ thấp đến cao.

- Counting Sort: đếm số lần xuất hiện của mỗi phần tử và sử dụng kết quả để xác định vị trí của các phần tử trong dãy.

- Bucket Sort: chia các phần tử vào các "xô" (bucket) và sắp xếp từng xô, cuối cùng kết hợp kết quả.

1. **Kết luận**

Các thuật toán sắp xếp có tư tưởng khác nhau tùy vào mục đích và tính chất của dữ liệu. Sắp xếp theo so sánh là phương pháp phổ biến nhất và có thể áp dụng cho bất kỳ kiểu dữ liệu nào, trong khi các thuật toán sắp xếp không so sánh như Counting Sort hay Radix Sort lại rất hiệu quả khi dữ liệu có giới hạn nhất định hoặc khi các phần tử có dạng số học dễ phân loại.

### 2. Thuật toán thích nhất ?Thuật toán không thích nhất ?

#### 2.1 Thuật toán thích nhất

Thuật toán thích nhất là Quick Sort vì đây là một trong những thuật toán sắp xếp có độ phức tạp trong trường hợp tốt nhất có thể đạt tới (O()). Cách hiểu cũng như là cách cài đặt không quá phức tạp. Trong các thư viện có sẵn của các trình biên dịch code, các hàm sort cũng được viết từ Quick Sort.

#### 2.2 Thuật toán không thích nhất ?

Thuật toán không thích nhất có thể là Merge Sort. Tuy đây là một trong những thuật toán sắp xếp nhanh nhất có độ phức tạp trong trường tốt nhất cũng là (O()), và cách hiểu cách hoạt động cũng không quá khó nhưng cách cài đặt là khá phức tạp, có thể nói là khó cài đặt nhất và nếu cài đặt sai có thể dẫn tới crack máy.

### 3. Trình bày và cài đặt các thuật toán sắp xếp

#### 3.1. **Sắp xếp nội (In-place Sorting)**

Thuật toán sắp xếp **In-place** là thuật toán không yêu cầu bộ nhớ bổ sung đáng kể ngoài bộ nhớ chứa dữ liệu ban đầu. Các thuật toán này thực hiện việc sắp xếp trực tiếp trên dãy dữ liệu mà không cần lưu trữ dữ liệu mới.

##### Các thuật toán In-place:

* **Interchange Sort**
* Bubble Sort
* **Selection Sort**
* **Insertion Sort**
* Binary Insertion Sort
* Shell Sort
* Shaker Sort
* **Quick Sort**
* **Heap Sort**

#### 3.2. **Sắp xếp ngoại (External Sorting)**

Thuật toán External Sorting thường được sử dụng khi dữ liệu quá lớn để lưu trữ trong bộ nhớ chính, chẳng hạn như trong các cơ sở dữ liệu lớn. Thuật toán này sắp xếp dữ liệu ngoài bộ nhớ và thường sử dụng các phương pháp phân tách, kết hợp các phần tử từ các tệp nhỏ.

Các thuật toán sắp xếp ngoại điển hình là:

* **Merge Sort**

#### 3.3 Cài đặt thuật toán

1. **Sắp xếp nội**
2. **Interchange Sort**

void interchangeSort(int\* arr, int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (arr[i] < arr[j]) {

                swap(arr[i], arr[j]);

            }

        }

    }

}

1. **Bubble Sort**

void bubbleSort(int\* arr, int n) {

    for (int i = 0; i < n - 1; i++) {

        bool swapped = false;

        for (int j = 0; j < n - i - 1; j++) {

            if (arr[j] < arr[j + 1]) {

                swap(arr[j], arr[j + 1]);

                swapped = true;

            }

        }

        if (!swapped) break;

    }

}

1. **Shaker Sort**

void shakerSort(int\* a, int n) {

    int up = 0, down = n - 1, hv = 0;

    while (up < down) {

        for (int i = up; i < down; i++) {

            if (a[i] < a[i + 1]) {

                swap(a[i], a[i + 1]);

                hv = i;

            }

        }

        down = hv;

        for (int j = down; j > up; j--) {

            if (a[j - 1] < a[j]) {

                swap(a[j - 1], a[j]);

                hv = j;

            }

        }

        up = hv;

    }

}

1. **Insertion Sort**

void insertionSort(int\* arr, int n) {

    for (int i = 1; i < n; i++) {

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] < key) {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

1. **Binary Insertion Sort**

int binarySearch(int\* arr, int item, int low, int high) {

    while (low <= high) {

        int mid = low + (high - low) / 2;

        if (arr[mid] > item)

            low = mid + 1;

        else

            high = mid - 1;

    }

    return low;

}

void binaryInsertionSort(int\* arr, int n) {

    for (int i = 1; i < n; i++) {

        int key = arr[i];

        int loc = binarySearch(arr, key, 0, i - 1);

        for (int j = i - 1; j >= loc; j--)

            arr[j + 1] = arr[j];

        arr[loc] = key;

    }

}

1. **Shell Sort**

void shellSort(int\* arr, int n) {

    for (int gap = n / 2; gap > 0; gap /= 2) { // Giảm dần gap

        for (int i = gap; i < n; i++) {

            int temp = arr[i];

            int j;

            for (j = i; j >= gap && arr[j - gap] < temp; j -= gap) {

                arr[j] = arr[j - gap];

            }

            arr[j] = temp;

        }

    }

}

1. **Selection Sort**

void selectionSort(int\* arr, int n) {

    for (int i = 0; i < n - 1; i++) {

        int max\_idx = i;

        for (int j = i + 1; j < n; j++)

            if (arr[j] > arr[max\_idx])

                max\_idx = j;

        swap(arr[i], arr[max\_idx]);

    }

}

1. **Heap Sort**

void heapify(int\* arr, int n, int i) {

    int largest = i, left = 2 \* i + 1, right = 2 \* i + 2;

    if (left < n && arr[left] > arr[largest]) largest = left;

    if (right < n && arr[right] > arr[largest]) largest = right;

    if (largest != i) {

        swap(arr[i], arr[largest]);

        heapify(arr, n, largest);

    }

}

void heapSort(int\* arr, int n) {

    for (int i = n / 2 - 1; i >= 0; i--) heapify(arr, n, i);

    for (int i = n - 1; i > 0; i--) {

        swap(arr[0], arr[i]);

        heapify(arr, i, 0);

    }

}

1. **Quick Sort** (Theo phân hoạch Lomuto)

int partition(int\* arr, int low, int high) {

    int pivot = arr[high];

    int i = low - 1;

    for (int j = low; j < high; j++) {

        if (arr[j] > pivot) swap(arr[++i], arr[j]);

    }

    swap(arr[i + 1], arr[high]);

    return i + 1;

}

void quickSort(int\* arr, int low, int high) {

    if (low < high) {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

1. **Sắp xếp ngoại**

* **Merge Sort**

void merge(int\* arr, int left, int mid, int right) {

    int n1 = mid - left + 1, n2 = right - mid;

    int\* L = new int[n1];

    int\* R = new int[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];

    for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2)

        arr[k++] = (L[i] >= R[j]) ? L[i++] : R[j++];

    while (i < n1) arr[k++] = L[i++];

    while (j < n2) arr[k++] = R[j++];

    delete[] L;

    delete[] R;

}

void mergeSort(int\* arr, int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);

        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

### 4. Ưu điểm và nhược điểm của các loại sort trên ? Cách khác phục những nhược điểm này là gì ?

#### 4.1 Interchange Sort

* Ưu điểm:
* Dễ hiểu và dễ cài đặt
* Nhược điểm:
* Hiệu suất kém với dữ liệu lớn, có độ phức tạp
* Sử dụng nhiều phép hoán đổi, làm giảm hiệu suất
* Cách khắc phục:
* Đây chỉ là thuật toán áp dụng trong giảng dạy nhằm cung cấp cho sinh viên kiến thúc về sắp xếp và cặp nghịch thế, thường không được áp dụng trong các dự án thực tế vì chi phí lớn, có thể dùng Quick Sort hoặc Merge Sort để thay thế

#### 4.2 Bubble Sort

* Ưu điểm:
* Dễ hiểu và dễ cài đặt
* Đối với các dãy gần như đã sắp xếp thì thuật toán sẽ kết thúc nhanh chóng
* Nhược điểm:
* Chi phí quá lớn đối với dữ liệu lớn, độ phức tạp
* Nhiều phép hoán đổi các đối ngẫu không cần thiết
* Cách khắc phục:
* Có thể tận dụng các thông tin ở mỗi lần duyệt để áp dụng vào lần duyệt sau, tránh duyệt lại các phần tử đã duyệt qua và không cần hoán đổi, đó là cơ chế của thuật toán cải tiến từ Bubble Sort có tên là Shaker Sort

#### 4.3 Shaker Sort

* Ưu điểm:
* Có các ưu điểm của Bubble Sort vì là thuật toán được cải tiến từ Bubble Sort
* Giảm bớt số bước thực hiện qua mỗi lần duyệt
* Nhược điểm:
* Tuy cải tiến những độ phức tạp trung bình vẫn là
* Không hiệu quả với dữ liệu lớn
* Cách khắc phục:
* Chuyển qua Quick Sort hoặc Merge Sort

#### 4.4 Insertion Sort

* Ưu điểm:
* Đơn giản, dễ hiểu và dễ cài đặt
* Hoạt động tốt với mảng nhỏ hoặc mảng gần như đã được sắp xếp
* Vì là In-place Sort nên không cần bộ nhớ ngoài
* Nhược điểm
* Độ phức t:ạp
* Cách khắc phục:
* Thuật toán sử dụng cơ chế tìm kiếm tuần tự, có thể thay đổi bằng tìm kiếm nhị phân để cải tiến thành Binary Insertion Sort hoặc một cơ chế khác là Shell Sort

#### 4.5 Binary Insertion Sort

* Ưu điểm:
* Giảm bớt số lần so sánh hơn Insertion Sort vì sử dụng tìm kiếm nhị phân
* Nhược điểm:
* Vẫn là độ phức tạp do các thao tác chuyển dịch phần tử
* Cách khắc phục:
* Sử dụng Merge Sort hoặc Quick Sort

#### 4.6 Shell Sort

* Ưu điểm:
* Cải tiến từ Insertion Sort bằng cách cho phép các phần tử di chuyển nhanh hơn thông qua các bước (gap)
* Cải thiện tốc độ xử lí hơn Insertion, đặc biệt trong các mảng lớn
* Nhược điểm:
* Phụ thuộc vào cách chọn bước (gap), không có một cách chọn tối ưu cho tất cả các trường hợp
* Cách khắc phục:
* Tìm cách cải tiến cách chọn bước nhảy sao cho tối tưu hoặc chuyển sang Quick Sort

#### 4.7 Selection Sort

* Ưu điểm:
* Đơn giản và dễ hiểu
* Số phép hoán đổi ít hơn Insertion và Bubble Sort
* Nhược điểm:
* Độ phức tạp vẫn là
* Không hiệu quả với mảng lớn
* Cách khắc phục:
* Dùng thay thế bằng Merge Sort hoặc Quick Sort

#### 4.8 Heap Sort

* Ưu điểm:
* Có độ phức tạp là ổn định và nhanh với mảng lớn
* Sử dụng bộ nhớ ngoài rất ít
* Nhược điểm:
* Sắp xếp không ổn định
* Các thao tác trên heap có thể khác phức tạp
* Khó hiểu và khó cài đặt đối với những người mới tiếp cận lập trình
* Cách khắc phục:
* Sử dụng Merge Sort nếu cần thêm sự ổn định

#### 4.9 Quick Sort

* Ưu điểm:
* Rất nhanh với độ phức tạp trung bình là
* Là thuật toán sắp xếp phổ biến đối với mảng lớn
* Nhược điểm:
* Có thể gặp hiệu suất kém khi phân vùng không đều (pivot kém)
* Không ổn định
* Cách khắc phục:
* Dùng phương pháp chọn pivot tốt hơn

#### 4.10 Merge Sort

* Ưu điểm:
* Độ phức tạp , ổn định và có hiểu quả với mảng lớn
* Sử dụng bộ nhớ ngoài, ít rò rỉ hơn Heap Sort
* Nhược điểm:
* Cần bộ nhớ phụ ngoài mảng ( bộ nhớ), không thích hợp với mảng rất lớn
* Không hiệu quả trong việc sắp xếp mảng nhỏ
* Khó cài đặt
* Cách khắc phục:
* Không sử dụng với các mảng nhỏ

#### 4.11 Radix Sort

* Ưu điểm:
* Không phụ thuộc vào các phép so sánh, có thể sắp xếp rất nhanh với dữ liệu phù hợp
* Độ phức tạp , rất có hiểu quả với các dữ liệu có số lượng phần tử nhỏ hoặc khi dữ liệu có kích thước đồng đều
* Nhược điểm:
* Chỉ sử dụng cho các loại dữ liệu số (hoặc chuỗi có độ dài cố định)
* Cần nhiều bộ nhớ phụ
* Cách khắc phục:
* Giới hạn việc sử dụng Radix Sort cho các trường hợp dữ liệu phù hợp, hoặc kết hợp với các thuật toán sắp xếp khác như QuickSort cho dữ liệu không thích hợp

## B. Bài tập cơ sở

### Bài 1:

a)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |

* Interchange Sort

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 5 | 39 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 1 | 39 | 8 | 5 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 1 | 8 | 39 | 5 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 1 | 5 | 39 | 8 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 1 | 3 | 39 | 8 | 5 | 6 | 9 | 12 | 4 | 7 | 10 |
| 1 | 3 | 8 | 39 | 5 | 6 | 9 | 12 | 4 | 7 | 10 |
| 1 | 3 | 5 | 39 | 8 | 6 | 9 | 12 | 4 | 7 | 10 |
| 1 | 3 | 4 | 39 | 8 | 6 | 9 | 12 | 5 | 7 | 10 |
| 1 | 3 | 4 | 8 | 39 | 6 | 9 | 12 | 5 | 7 | 10 |
| 1 | 3 | 4 | 6 | 39 | 8 | 9 | 12 | 5 | 7 | 10 |
| 1 | 3 | 4 | 5 | 39 | 8 | 9 | 12 | 6 | 7 | 10 |
| 1 | 3 | 4 | 5 | 8 | 39 | 9 | 12 | 6 | 7 | 10 |
| 1 | 3 | 4 | 5 | 6 | 39 | 9 | 12 | 8 | 7 | 10 |
| 1 | 3 | 4 | 5 | 6 | 9 | 39 | 12 | 8 | 7 | 10 |
| 1 | 3 | 4 | 5 | 6 | 8 | 39 | 12 | 9 | 7 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 39 | 12 | 9 | 8 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 12 | 39 | 9 | 8 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 9 | 39 | 12 | 8 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 39 | 12 | 9 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 12 | 39 | 9 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 39 | 12 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 12 | 39 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 39 | 12 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 39 |

* Bubble Sort

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 5 | 39 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 5 | 8 | 39 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 5 | 8 | 1 | 39 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 5 | 8 | 1 | 3 | 39 | 6 | 9 | 12 | 4 | 7 | 10 |
| 5 | 8 | 1 | 3 | 6 | 39 | 9 | 12 | 4 | 7 | 10 |
| 5 | 8 | 1 | 3 | 6 | 9 | 39 | 12 | 4 | 7 | 10 |
| 5 | 8 | 1 | 3 | 6 | 9 | 12 | 39 | 4 | 7 | 10 |
| 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 39 | 7 | 10 |
| 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 39 | 10 |
| 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 39 |
| 5 | 8 | 1 | 3 | 6 | 9 | 4 | 12 | 7 | 10 | 39 |
| 5 | 8 | 1 | 3 | 6 | 4 | 9 | 12 | 7 | 10 | 39 |
| 5 | 8 | 1 | 3 | 4 | 6 | 9 | 12 | 7 | 10 | 39 |
| 5 | 1 | 8 | 3 | 4 | 6 | 9 | 12 | 7 | 10 | 39 |
| 1 | 5 | 8 | 3 | 4 | 6 | 9 | 12 | 7 | 10 | 39 |
| 1 | 5 | 3 | 8 | 4 | 6 | 9 | 12 | 7 | 10 | 39 |
| 1 | 5 | 3 | 4 | 8 | 6 | 9 | 12 | 7 | 10 | 39 |
| 1 | 5 | 3 | 4 | 6 | 8 | 9 | 12 | 7 | 10 | 39 |
| 1 | 5 | 3 | 4 | 6 | 8 | 9 | 7 | 12 | 10 | 39 |
| 1 | 5 | 3 | 4 | 6 | 8 | 9 | 7 | 10 | 12 | 39 |
| 1 | 5 | 3 | 4 | 6 | 8 | 7 | 9 | 10 | 12 | 39 |
| 1 | 5 | 3 | 4 | 6 | 7 | 8 | 9 | 10 | 12 | 39 |
| 1 | 3 | 5 | 4 | 6 | 7 | 8 | 9 | 10 | 12 | 39 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 39 |

* Insertion Sort

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | key |
| 5 | 39 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 5 |
| 5 | 8 | 39 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 8 |
| 5 | 8 |  | 39 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 1 |
| 5 |  | 8 | 39 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 1 |
| 1 | 5 | 8 | 39 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 1 |
| 1 | 5 | 8 |  | 39 | 6 | 9 | 12 | 4 | 7 | 10 | 3 |
| 1 | 5 |  | 8 | 39 | 6 | 9 | 12 | 4 | 7 | 10 | 3 |
| 1 | 3 | 5 | 8 | 39 | 6 | 9 | 12 | 4 | 7 | 10 | 3 |
| 1 | 3 | 5 | 8 |  | 39 | 9 | 12 | 4 | 7 | 10 | 6 |
| 1 | 3 | 5 | 6 | 8 | 39 | 9 | 12 | 4 | 7 | 10 | 6 |
| 1 | 3 | 5 | 6 | 8 | 9 | 39 | 12 | 4 | 7 | 10 | 9 |
| 1 | 3 | 5 | 6 | 8 | 9 | 12 | 39 | 4 | 7 | 10 | 12 |
| 1 | 3 | 5 | 6 | 8 | 9 | 12 |  | 39 | 7 | 10 | 4 |
| 1 | 3 | 5 | 6 | 8 | 9 |  | 12 | 39 | 7 | 10 | 4 |
| 1 | 3 | 5 | 6 | 8 |  | 9 | 12 | 39 | 7 | 10 | 4 |
| 1 | 3 | 5 | 6 |  | 8 | 9 | 12 | 39 | 7 | 10 | 4 |
| 1 | 3 | 5 |  | 6 | 8 | 9 | 12 | 39 | 7 | 10 | 4 |
| 1 | 3 | 4 | 5 | 6 | 8 | 9 | 12 | 39 | 7 | 10 | 4 |
| 1 | 3 | 4 | 5 | 6 | 8 | 9 | 12 |  | 39 | 10 | 7 |
| 1 | 3 | 4 | 5 | 6 | 8 | 9 |  | 12 | 39 | 10 | 7 |
| 1 | 3 | 4 | 5 | 6 | 8 |  | 9 | 12 | 39 | 10 | 7 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 12 | 39 | 10 | 7 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 12 |  | 39 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 39 | 10 |

* Selection Sort

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  |
| 39 | 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | min |
| 39 | 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 0 |
| 1 | 5 | 8 | 39 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 3 |
| 1 | 5 | 8 | 39 | 3 | 6 | 9 | 12 | 4 | 7 | 10 | 1 |
| 1 | 3 | 8 | 39 | 5 | 6 | 9 | 12 | 4 | 7 | 10 | 4 |
| 1 | 3 | 8 | 39 | 5 | 6 | 9 | 12 | 4 | 7 | 10 | 2 |
| 1 | 3 | 4 | 39 | 5 | 6 | 9 | 12 | 8 | 7 | 10 | 8 |
| 1 | 3 | 4 | 39 | 5 | 6 | 9 | 12 | 8 | 7 | 10 | 3 |
| 1 | 3 | 4 | 5 | 39 | 6 | 9 | 12 | 8 | 7 | 10 | 4 |
| 1 | 3 | 4 | 5 | 39 | 6 | 9 | 12 | 8 | 7 | 10 | 4 |
| 1 | 3 | 4 | 5 | 6 | 39 | 9 | 12 | 8 | 7 | 10 | 5 |
| 1 | 3 | 4 | 5 | 6 | 39 | 9 | 12 | 8 | 7 | 10 | 5 |
| 1 | 3 | 4 | 5 | 6 | 7 | 9 | 12 | 8 | 39 | 10 | 9 |
| 1 | 3 | 4 | 5 | 6 | 7 | 9 | 12 | 8 | 39 | 10 | 6 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 12 | 9 | 39 | 10 | 8 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 12 | 9 | 39 | 10 | 7 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 12 | 39 | 10 | 8 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 12 | 39 | 10 | 8 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 39 | 12 | 10 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 39 | 12 | 9 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 | 39 | 10 |

b)

* Interchange Sort

#include <iostream>

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

void interchangeSort(int arr[], int n)

{

    for (int i = 0; i < n - 1; i++)

    {

        for (int j = i + 1; j < n; j++)

        {

            if (arr[i] > arr[j])

            {

                swap(arr[i], arr[j]);

            }

        }

    }

}

void printArrar(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        std::cout << arr[i] << " ";

    }

}

int main()

{

    int arr[] = {39, 5, 8, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    interchangeSort(arr, n);

    printArrar(arr, n);

    return 0;

}

* Bubble Sort

#include <iostream>

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

void bubbleSort(int arr[], int n)

{

    int up = 0, down = n - 1;

    while (up < down)

    {

        for (int i = up; i < down; i++)

        {

            if (arr[i] > arr[i + 1])

            {

                swap(arr[i], arr[i + 1]);

            }

        }

        down--;

        for (int i = down; i > up; i--)

        {

            if (arr[i - 1] > arr[i])

            {

                swap(arr[i - 1], arr[i]);

            }

        }

        up++;

    }

}

void printArrar(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        std::cout << arr[i] << " ";

    }

}

int main()

{

    int arr[] = {39, 5, 8, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    bubbleSort(arr, n);

    printArrar(arr, n);

    return 0;

}

* Insertion Sort

#include <iostream>

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

void insertionSort(int arr[], int n)

{

    for (int i = 1; i < n; ++i)

    {

        int key = arr[i];

        int j = i - 1;

        while (arr[j] > key && j >= 0)

        {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

void printArrar(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        std::cout << arr[i] << " ";

    }

}

int main()

{

    int arr[] = {39, 5, 8, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);

    printArrar(arr, n);

    return 0;

}

* Selection Sort

#include <iostream>

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

void selectionSort(int arr[], int n)

{

    for (int i = 0; i < n - 1; ++i)

    {

        int min = i;

        for (int j = i + 1; j < n; ++j)

        {

            if (arr[min] > arr[j])

            {

                min = j;

            }

        }

        swap(arr[min], arr[i]);

    }

}

void printArrar(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        std::cout << arr[i] << " ";

    }

}

int main()

{

    int arr[] = {39, 5, 8, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    selectionSort(arr, n);

    printArrar(arr, n);

    return 0;

}

c)

* Interchange Sort

for (int i = 0; i < n - 1; i++)

- chạy từ i = 0 đến i = n – 2, tức là i sẽ chạy (n-1) lần.

for (int j = i + 1; j < n; j++)

        {

            if (arr[i] > arr[j])

            {

                swap(arr[i], arr[j]);

            }

        }

- chạy từ j = i + 1 đến j = n – 1, tức là j sẽ chạy (n – i – 1) lần.

- bên trong vòng lập trong (với biến j) số phép so sanh được thực hiện là:

* Khi i = 0 🡪 vòng lặp trong chạy (n – 1) lần, tức là có (n – 1) phép so sánh
* Khi i = 1 🡪 vòng lặp trong chạy (n – 2) lần, tức là có (n – 2) phép so sánh
* Khi i = 2 🡪 vòng lặp trong chạy (n – 3) lần, tức là có (n – 3) phép so sánh
* …..
* Khi i = n – 2 🡪 vòng lặp trong chạy 1 lần, tức là có 1 phép so sánh

🡪 Vậy tổng số phép so sánh được thực hiện là

hay độ phức tạp trong trường hợp xấu nhất sẽ là .

(mỗi phép so sánh tốn thời gian nên tổng số phép so sánh chính là độ phức tạp của thời gian)

* Bubble Sort

while (up < down)

- up chạy từ 0 và down chạy từ (n – 1), vòng lặp kết thúc khi up ≥ down. Mỗi lần, up tăng lên 1 và down lại giảm đi 1 🡪 mỗi lần lặp phạm vi của các phần tử sẽ thu hẹp lại.

🡪 vòng lặp này sẽ mất khoảng lần, tức là về mặt phức tạp của thời gian.

for (int i = up; i < down; i++)

        {

            if (arr[i] > arr[i + 1])

            {

                swap(arr[i], arr[i + 1]);

            }

        }

        down--;

- vòng lặp bắt đầu với i = up và kết thúc khi i = down – 1, và thực hiện 1 phép so sánh (nếu đúng sẽ đổi chỗ) mỗi lần duyệt 🡪 có (down – up) phép so sánh.

- mà up bắt đầu tăng từ 0 và down bắt đầu giảm (n – 1) 🡪 có (n – 1) phép so sánh, tức là có phép so sánh cho đến khi vòng lặp while kết thúc.

for (int i = down; i > up; i--)

        {

            if (arr[i - 1] > arr[i])

            {

                swap(arr[i - 1], arr[i]);

            }

        }

        up++;

- tương tự tổng số phép so sánh là .

🡪 Vậy tổng phép so sánh cho 2 vòng lặp con là hay độ phức tạp trong trường hợp xấu nhất là .

* Insertion Sort

for (int i = 1; i < n; ++i)

- chạy từ i = 1 đến i = n – 1, tức là i sẽ chạy (n-1) lần

int key = arr[i];

        int j = i - 1;

        while (arr[j] > key && j >= 0)

        {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

- với mỗi phép gán sẽ mất thời gian 🡪 (n – 1) phép gán tương tự sẽ có thời gian.

- Trong vòng lặp while, phép so sánh và hoán đổi sẽ bắt đầu từ (i – 1) cho đến đầu mảng 🡪 trong trường hợp xấu nhất, phần tử cuối cùng sẽ được dịch chuyển tới đầu mảng, tức vòng lặp trong phải chạy i lần với i là chỉ số vòng lặp ngoài.

- Khi đó:

* i = 1: vòng lặp trong có thể chạy 1 lần
* i = 2: vòng lặp trong có thể chạy 2 lần
* i = 3: vòng lặp trong có thể chạy 3 lần
* …
* i = n – 1: vòng lặp trong có thể chạy (n – 1) lần

🡪 Tổng số phép so sánh và hoán đổi là:

hay có thể nói thuật toán có độ phức tạp là .

* Selection Sort

for (int i = 0; i < n - 1; ++i)

- chạy từ i = 0 đến i = n – 2, tức là i sẽ chạy (n-1) lần

    int min = i;

        for (int j = i + 1; j < n; ++j)

        {

            if (arr[min] > arr[j])

            {

                min = j;

            }

        }

        if (min != i)

            swap(arr[min], arr[i]);

- chạy từ j = i + 1 đến j = n – 1, tức là j sẽ chạy (n – i – 1) lần.

- bên trong vòng lập trong (với biến j) số phép so sanh được thực hiện là:

* Khi i = 0 🡪 vòng lặp trong chạy (n – 1) lần, tức là có (n – 1) phép so sánh
* Khi i = 1 🡪 vòng lặp trong chạy (n – 2) lần, tức là có (n – 2) phép so sánh
* Khi i = 2 🡪 vòng lặp trong chạy (n – 3) lần, tức là có (n – 3) phép so sánh
* …..
* Khi i = n – 2 🡪 vòng lặp trong chạy 1 lần, tức là có 1 phép so sánh

🡪 Vậy tổng số phép so sánh được thực hiện là

- phép so sánh và hoán đổi sau vòng lặp biến j là không đáng kể và có thể ≤ (n – 1) nên độ phức tạp của thuật toán chính là phép tổng số phép so sánh trong vòng lập biến j, tức là.

### Bài 2:

a)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |

* Quick Sort (Lomuto)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 4 | 12 | 7 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 4 | 7 | 12 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 4 | 7 | 10 | 12 |
| 5 | 8 | 1 | 3 | 6 | 9 | 4 | 7 | 10 | 12 |
| 5 | 1 | 8 | 3 | 6 | 9 | 4 | 7 | 10 | 12 |
| 5 | 1 | 3 | 8 | 6 | 9 | 4 | 7 | 10 | 12 |
| 5 | 1 | 3 | 6 | 8 | 9 | 4 | 7 | 10 | 12 |
| 5 | 1 | 3 | 6 | 4 | 9 | 8 | 7 | 10 | 12 |
| 5 | 1 | 3 | 6 | 4 | 7 | 8 | 9 | 10 | 12 |
| 1 | 5 | 3 | 6 | 4 | 7 | 8 | 9 | 10 | 12 |
| 1 | 3 | 5 | 6 | 4 | 7 | 8 | 9 | 10 | 12 |
| 1 | 3 | 4 | 6 | 5 | 7 | 8 | 9 | 10 | 12 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 |

|  |  |
| --- | --- |
|  | pivot |
|  | swap |

* Merge Sort

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 5 | 8 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 1 | 3 | 5 | 6 | 8 | 4 | 7 | 9 | 10 | 12 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 |

* Heap Sort

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 5 | 1 | 3 | 6 | 9 | 12 | 4 | 7 | 10 |
| 8 | 5 | 1 | 3 | 10 | 9 | 12 | 4 | 7 | 6 |
| 8 | 5 | 1 | 7 | 10 | 9 | 12 | 4 | 3 | 6 |
| 8 | 5 | 12 | 7 | 10 | 9 | 1 | 4 | 3 | 6 |
| 8 | 10 | 12 | 7 | 5 | 9 | 1 | 4 | 3 | 6 |
| 12 | 10 | 8 | 7 | 5 | 9 | 1 | 4 | 3 | 6 |
| 6 | 10 | 8 | 7 | 5 | 9 | 1 | 4 | 3 | 12 |
| 10 | 6 | 8 | 7 | 5 | 9 | 1 | 4 | 3 | 12 |
| 10 | 7 | 8 | 6 | 5 | 9 | 1 | 4 | 3 | 12 |
| 10 | 7 | 9 | 6 | 5 | 8 | 1 | 4 | 3 | 12 |
| 3 | 7 | 9 | 6 | 5 | 8 | 1 | 4 | 10 | 12 |
| 9 | 7 | 3 | 6 | 5 | 8 | 1 | 4 | 10 | 12 |
| 9 | 7 | 8 | 6 | 5 | 3 | 1 | 4 | 10 | 12 |
| 4 | 7 | 8 | 6 | 5 | 3 | 1 | 9 | 10 | 12 |
| 8 | 7 | 4 | 6 | 5 | 3 | 1 | 9 | 10 | 12 |
| 1 | 7 | 4 | 6 | 5 | 3 | 8 | 9 | 10 | 12 |
| 7 | 1 | 4 | 6 | 5 | 3 | 8 | 9 | 10 | 12 |
| 7 | 6 | 4 | 1 | 5 | 3 | 8 | 9 | 10 | 12 |
| 3 | 6 | 4 | 1 | 5 | 7 | 8 | 9 | 10 | 12 |
| 6 | 3 | 4 | 1 | 5 | 7 | 8 | 9 | 10 | 12 |
| 6 | 5 | 4 | 1 | 3 | 7 | 8 | 9 | 10 | 12 |
| 3 | 5 | 4 | 1 | 6 | 7 | 8 | 9 | 10 | 12 |
| 5 | 3 | 4 | 1 | 6 | 7 | 8 | 9 | 10 | 12 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 |
| 4 | 3 | 1 | 5 | 6 | 7 | 8 | 9 | 10 | 12 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 |
| 3 | 1 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 12 |

b)

* Quick Sort (Lomuto)

#include <iostream>

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

int partition(int arr[], int low, int high)

{

    int pivot = arr[high];

    int i = low - 1;

    for (int j = low; j < high; ++j)

    {

        if (arr[j] < pivot)

        {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]);

    return i + 1;

}

void quickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

void printArrar(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        std::cout << arr[i] << " ";

    }

}

int main()

{

    int arr[] = {8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n - 1);

    printArrar(arr, n);

    return 0;

}

* Merge Sort

#include <iostream>

void merge(int arr[], int l, int m, int r)

{

    int n1 = m - l + 1;

    int n2 = r - m;

    int L[n1], R[n2];

    for (int i = 0; i < n1; ++i)

    {

        L[i] = arr[l + i];

    }

    for (int i = 0; i < n2; ++i)

    {

        R[i] = arr[m + 1 + i];

    }

    int 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++;

    }

}

void mergeSort(int arr[], int l, int r)

{

    if (l >= r)

    {

        return;

    }

    int m = l + (r - l) / 2;

    mergeSort(arr, l, m);

    mergeSort(arr, m + 1, r);

    merge(arr, l, m, r);

}

void printArrar(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        std::cout << arr[i] << " ";

    }

}

int main()

{

    int arr[] = {39, 5, 8, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    mergeSort(arr, 0, n - 1);

    printArrar(arr, n);

    return 0;

}

* Heap Sort

#include <iostream>

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

void heapify(int arr[], int n, int i)

{

    int largest = i;

    int l = 2 \* i + 1;

    int r = 2 \* i + 2;

    if (l < n && arr[l] > arr[largest])

    {

        largest = l;

    }

    if (r < n && arr[r] > arr[largest])

    {

        largest = r;

    }

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        heapify(arr, n, largest);

    }

}

void heapSort(int arr[], int n)

{

    for (int i = n / 2 - 1; i >= 0; --i)

    {

        heapify(arr, n, i);

    }

    for (int i = n - 1; i > 0; --i)

    {

        swap(arr[0], arr[i]);

        heapify(arr, i, 0);

    }

}

void printArrar(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        std::cout << arr[i] << " ";

    }

}

int main()

{

    int arr[] = {39, 5, 8, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    printArrar(arr, n);

    return 0;

}

c)

* Quick Sort

- Quick Sort cũng giống với Merge Sort đều dùng phương pháp chia để trị nhưng lại khác Merge Sort ở chỗ Quick Sort là dạng trị rồi mới chia, tức là thay vì cứ tìm điểm mid của mảng để chia như Merge Sort thì Quick Sort sẽ chia bằng cách tìm điểm pivot 🡪 và đối với trường hợp tốt nhất (best case) thì điểm pivot đó cũng chính là điểm mid trong Merge Sort.

void quickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1); (1)

        quickSort(arr, pi + 1, high); (2)

    }

}

- Gọi là thời gian thực hiện của thuật toán với n là chiều dài của mảng, ở (1) và (2), sẽ đệ quy về dãy có chiều dài bằng ½ dãy cha, cứ như v cho đến khác các phần tử của mảng ban đầu trở thành n mảng riêng biệt, v mỗi lần đệ quy dãy con sẽ có thời gian thực hiện, do dãy con có chiều dài bằng ½ dãy cha, và sau đó hợp các mảng con lại mất n bước, từ đây ta có công thức:

Dãy con tiếp tục đệ quy:

Ta thấy rằng mỗi bước đệ quy ở dãy con 🡪 ta sẽ cần n bước để “nối lại”

Từ đây có công thức tổng quát cho trường hợp trung bình

Với k là số lần đệ quy, v với 🡪 công thức trở thành

Vậy thuật toán có độ phức tạp thời gian là .

- Nhưng khác với Merge Sort thì Quick Sort lại không có tính ổn định do điểm pivot không phải lúc nào cũng là điểm mid, và giả sử điẻm pivot là phần tử có giá trị nhỏ nhất hoặc lớn nhất thì nó sẽ chia mảng làm 2 phần trên lệch rất nhiều. Tệ nhất là 1 phần một có (n-1) phần tử, trong khi đó phần còn lại là 0 phần tử, ta có

Tiếp tục đệ quy ở các dãy con, ta có:

Từ đây ta có công thức tổng quát:

Vậy đối với Quick Sort thì nếu là Worst Case thì độ phức tạp thuật toán cũng sẽ là .

* Merge Sort

void mergeSort(int arr[], int l, int r)

{

    if (l >= r)

    {

        return;

    }

    int m = l + (r - l) / 2;

    mergeSort(arr, l, m); (1)

    mergeSort(arr, m + 1, r); (2)

    merge(arr, l, m, r);

}

- Gọi là thời gian thực hiện của thuật toán với n là chiều dài của mảng, ở (1) và (2), sẽ đệ quy về dãy có chiều dài bằng ½ dãy cha, cứ như v cho đến khác các phần tử của mảng ban đầu trở thành n mảng riêng biệt, v mỗi lần đệ quy dãy con sẽ có thời gian thực hiện, do dãy con có chiều dài bằng ½ dãy cha, và sau đó hợp các mảng con lại mất n bước, từ đây ta có công thức:

Dãy con tiếp tục đệ quy:

Ta thấy rằng mỗi bước đệ quy ở dãy con 🡪 ta sẽ cần n bước để “nối lại”

Từ đây có công thức tổng quát cho trường hợp trung bình

Với k là số lần đệ quy, v với 🡪 công thức trở thành

Vậy thuật toán có độ phức tạp thời gian là .

* Heap Sort

- Heap Sort được chia ra làm 2 giai đoạn:

* Build max-heap
* Sorting

- Gọi là thời gian thực hiện thuật toán với n là chiều dài hay số phần tử của mảng.

void heapify(int arr[], int n, int i)

{

    int largest = i;

    int l = 2 \* i + 1;

    int r = 2 \* i + 2;

    if (l < n && arr[l] > arr[largest])

    {

        largest = l;

    }

    if (r < n && arr[r] > arr[largest])

    {

        largest = r;

    }

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        heapify(arr, n, largest); (1)

    }

}

void heapSort(int arr[], int n)

{

    for (int i = n / 2 - 1; i >= 0; --i)

    {

        heapify(arr, n, i); (2)

    }

    for (int i = n - 1; i > 0; --i)

    {

        swap(arr[0], arr[i]);

        heapify(arr, i, 0); (3)

    }

}

- Ta bắt đầu bằng việc heapify từ vị trí (n/2 – 1) về 0. Mỗi lần heapify, độ phức tạp sẽ phụ thuộc vào chiều cao của cây con mà heapify đang xử lý.

* Nếu heapify được gọi cho một cây con ở mức cao của cây thì độ phức tạp sẽ là do chiều cao tối đa của cây nhị phân là .
* Nếu heapify được gọi cho một cây con gần đáy của cây, nó sẽ yêu cầu ít thời gian hơn, có thể chỉ mất , vì các cây con này có chiều cao thấp.

- Bây giờ, chúng ta sẽ tính tổng độ phức tạp của việc gọi heapify cho toàn bộ cây, từ phần tử gần đáy đến phần tử gốc (từ chỉ số n/2−1 xuống 0).

* Cây nhị phân có n phần tử. Ở mức độ sâu h (mức cao trong cây), có khoảng nút. Chiều cao của cây là .
* Độ phức tạp của một lần gọi heapify cho một nút là , với là chiều cao của cây con đó.

- Bây giờ, chúng ta sẽ tính tổng độ phức tạp của tất cả các lần gọi heapify:

* Tại mức (mức gốc), có 1 nút, độ phức tạp là .
* Tại mức , có 2 nút, độ phức tạp mỗi nút là .
* Tại mức, có 4 nút, độ phức tạp mỗi nút là .
* Cứ như vậy, các mức dưới cây sẽ yêu cầu ít thời gian hơn.

- Từ đây ta có công thức tổng quát cho cả quá trình:

- Vì có giá trị không quá lớn so với giá trị của n, nên có độ phức tạp là

Vậy t có độ phức tạp của thuật toán là .

### Bài 3:

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

// Interchange Sort (Traditional Bubble Sort)

void interchangeSort(int arr[], int n, int &comparisons, int &swaps)

{

    for (int i = 0; i < n - 1; i++)

    {

        for (int j = i + 1; j < n; j++)

        {

            comparisons++;

            if (arr[i] > arr[j])

            {

                swap(arr[i], arr[j]);

                swaps++;

            }

        }

    }

}

// Bubble Sort

void bubbleSort(int arr[], int n, int &comparisons, int &swaps)

{

    int up = 0, down = n - 1;

    while (up < down)

    {

        for (int i = up; i < down; i++)

        {

            comparisons++;

            if (arr[i] > arr[i + 1])

            {

                swap(arr[i], arr[i + 1]);

                swaps++;

            }

        }

        down--;

        for (int i = down; i > up; i--)

        {

            comparisons++;

            if (arr[i - 1] > arr[i])

            {

                swap(arr[i - 1], arr[i]);

                swaps++;

            }

        }

        up++;

    }

}

// Shaker Sort (Cocktail Sort)

void shakerSort(int arr[], int n, int &comparisons, int &swaps)

{

    int up = 0, down = n - 1;

    while (up < down)

    {

        for (int i = up; i < down; i++)

        {

            comparisons++;

            if (arr[i] > arr[i + 1])

            {

                swap(arr[i], arr[i + 1]);

                swaps++;

            }

        }

        down--;

        for (int i = down; i > up; i--)

        {

            comparisons++;

            if (arr[i - 1] > arr[i])

            {

                swap(arr[i - 1], arr[i]);

                swaps++;

            }

        }

        up++;

    }

}

// Insertion Sort

void insertionSort(int arr[], int n, int &comparisons, int &assignments)

{

    for (int i = 1; i < n; i++)

    {

        int key = arr[i];

        int j = i - 1;

        comparisons++;

        while (j >= 0 && arr[j] > key)

        {

            arr[j + 1] = arr[j];

            assignments++;

            j--;

            comparisons++;

        }

        arr[j + 1] = key;

        assignments++;

    }

}

// Binary Insertion Sort

void binaryInsertionSort(int arr[], int n, int &comparisons, int &assignments)

{

    for (int i = 1; i < n; i++)

    {

        int key = arr[i];

        int left = 0, right = i - 1;

        while (left <= right)

        {

            comparisons++;

            int mid = left + (right - left) / 2;

            if (arr[mid] < key)

            {

                left = mid + 1;

            }

            else

            {

                right = mid - 1;

            }

        }

        for (int j = i - 1; j >= left; j--)

        {

            arr[j + 1] = arr[j];

            assignments++;

        }

        arr[left] = key;

        assignments++;

    }

}

// Selection Sort

void selectionSort(int arr[], int n, int &comparisons, int &swaps)

{

    for (int i = 0; i < n - 1; i++)

    {

        int minIdx = i;

        for (int j = i + 1; j < n; j++)

        {

            comparisons++;

            if (arr[j] < arr[minIdx])

            {

                minIdx = j;

            }

        }

        if (minIdx != i)

        {

            swap(arr[i], arr[minIdx]);

            swaps++;

        }

    }

}

// Shell Sort

void shellSort(int arr[], int n, int &comparisons, int &swaps)

{

    for (int gap = n / 2; gap > 0; gap /= 2)

    {

        for (int i = gap; i < n; i++)

        {

            int temp = arr[i];

            int j = i;

            comparisons++;

            while (j >= gap && arr[j - gap] > temp)

            {

                arr[j] = arr[j - gap];

                swaps++;

                j -= gap;

                comparisons++;

            }

            arr[j] = temp;

        }

    }

}

// Heap Sort

void heapify(int arr[], int n, int i, int &comparisons, int &swaps)

{

    int largest = i;

    int l = 2 \* i + 1;

    int r = 2 \* i + 2;

    if (l < n && arr[l] > arr[largest])

    {

        largest = l;

    }

    comparisons++;

    if (r < n && arr[r] > arr[largest])

    {

        largest = r;

    }

    comparisons++;

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        swaps++;

        heapify(arr, n, largest, comparisons, swaps);

    }

}

void heapSort(int arr[], int n, int &comparisons, int &swaps)

{

    for (int i = n / 2 - 1; i >= 0; i--)

    {

        heapify(arr, n, i, comparisons, swaps);

    }

    for (int i = n - 1; i > 0; i--)

    {

        swap(arr[0], arr[i]);

        swaps++;

        heapify(arr, i, 0, comparisons, swaps);

    }

}

// Quick Sort

int partition(int arr[], int low, int high, int &comparisons, int &swaps)

{

    int pivot = arr[high];

    int i = low - 1;

    for (int j = low; j < high; j++)

    {

        comparisons++;

        if (arr[j] < pivot)

        {

            i++;

            swap(arr[i], arr[j]);

            swaps++;

        }

    }

    swap(arr[i + 1], arr[high]);

    swaps++;

    return i + 1;

}

void quickSort(int arr[], int low, int high, int &comparisons, int &swaps)

{

    if (low < high)

    {

        int pi = partition(arr, low, high, comparisons, swaps);

        quickSort(arr, low, pi - 1, comparisons, swaps);

        quickSort(arr, pi + 1, high, comparisons, swaps);

    }

}

// Merge Sort

void merge(int arr[], int left, int mid, int right, int &comparisons, int &swaps)

{

    int n1 = mid - left + 1;

    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)

    {

        L[i] = arr[left + i];

    }

    for (int i = 0; i < n2; i++)

    {

        R[i] = arr[mid + 1 + i];

    }

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2)

    {

        comparisons++;

        if (L[i] <= R[j])

        {

            arr[k] = L[i];

            i++;

        }

        else

        {

            arr[k] = R[j];

            j++;

        }

        swaps++;

        k++;

    }

    while (i < n1)

    {

        arr[k] = L[i];

        i++;

        k++;

        swaps++;

    }

    while (j < n2)

    {

        arr[k] = R[j];

        j++;

        k++;

        swaps++;

    }

}

void mergeSort(int arr[], int left, int right, int &comparisons, int &swaps)

{

    if (left < right)

    {

        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid, comparisons, swaps);

        mergeSort(arr, mid + 1, right, comparisons, swaps);

        merge(arr, left, mid, right, comparisons, swaps);

    }

}

// Generate random array

void generateRandomArray(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        arr[i] = rand() % 10000; // Random number between 0 and 9999

    }

}

int main()

{

    srand(time(0));

    int n\_values[] = {10, 100, 200, 500, 1000, 2000, 5000, 10000};

    int num\_trials = 10;

    for (int i = 0; i < 8; i++)

    {

        int n = n\_values[i];

        cout << "For n = " << n << ":\n";

        int total\_comparisons\_interchange = 0, total\_swaps\_interchange = 0;

        int total\_comparisons\_bubble = 0, total\_swaps\_bubble = 0;

        int total\_comparisons\_shaker = 0, total\_swaps\_shaker = 0;

        int total\_comparisons\_insertion = 0, total\_assignments\_insertion = 0;

        int total\_comparisons\_binary\_insertion = 0, total\_assignments\_binary\_insertion = 0;

        int total\_comparisons\_selection = 0, total\_swaps\_selection = 0;

        int total\_comparisons\_shell = 0, total\_swaps\_shell = 0;

        int total\_comparisons\_heap = 0, total\_swaps\_heap = 0;

        int total\_comparisons\_quick = 0, total\_swaps\_quick = 0;

        int total\_comparisons\_merge = 0, total\_swaps\_merge = 0;

        for (int j = 0; j < num\_trials; j++)

        {

            int arr\_interchange[n], arr\_bubble[n], arr\_shaker[n], arr\_insertion[n], arr\_binary\_insertion[n], arr\_selection[n], arr\_shell[n], arr\_heap[n], arr\_quick[n], arr\_merge[n];

            generateRandomArray(arr\_interchange, n);

            generateRandomArray(arr\_bubble, n);

            generateRandomArray(arr\_shaker, n);

            generateRandomArray(arr\_insertion, n);

            generateRandomArray(arr\_binary\_insertion, n);

            generateRandomArray(arr\_selection, n);

            generateRandomArray(arr\_shell, n);

            generateRandomArray(arr\_heap, n);

            generateRandomArray(arr\_quick, n);

            generateRandomArray(arr\_merge, n);

            // Interchange Sort

            int comparisons = 0, swaps = 0;

            interchangeSort(arr\_interchange, n, comparisons, swaps);

            total\_comparisons\_interchange += comparisons;

            total\_swaps\_interchange += swaps;

            // Bubble Sort

            comparisons = 0;

            swaps = 0;

            bubbleSort(arr\_bubble, n, comparisons, swaps);

            total\_comparisons\_bubble += comparisons;

            total\_swaps\_bubble += swaps;

            // Shaker Sort

            comparisons = 0;

            swaps = 0;

            shakerSort(arr\_shaker, n, comparisons, swaps);

            total\_comparisons\_shaker += comparisons;

            total\_swaps\_shaker += swaps;

            // Insertion Sort

            comparisons = 0;

            swaps = 0;

            insertionSort(arr\_insertion, n, comparisons, swaps);

            total\_comparisons\_insertion += comparisons;

            total\_assignments\_insertion += swaps;

            // Binary Insertion Sort

            comparisons = 0;

            swaps = 0;

            binaryInsertionSort(arr\_binary\_insertion, n, comparisons, swaps);

            total\_comparisons\_binary\_insertion += comparisons;

            total\_assignments\_binary\_insertion += swaps;

            // Selection Sort

            comparisons = 0;

            swaps = 0;

            selectionSort(arr\_selection, n, comparisons, swaps);

            total\_comparisons\_selection += comparisons;

            total\_swaps\_selection += swaps;

            // Shell Sort

            comparisons = 0;

            swaps = 0;

            shellSort(arr\_shell, n, comparisons, swaps);

            total\_comparisons\_shell += comparisons;

            total\_swaps\_shell += swaps;

            // Heap Sort

            comparisons = 0;

            swaps = 0;

            heapSort(arr\_heap, n, comparisons, swaps);

            total\_comparisons\_heap += comparisons;

            total\_swaps\_heap += swaps;

            // Quick Sort

            comparisons = 0;

            swaps = 0;

            quickSort(arr\_quick, 0, n - 1, comparisons, swaps);

            total\_comparisons\_quick += comparisons;

            total\_swaps\_quick += swaps;

            // Merge Sort

            comparisons = 0;

            swaps = 0;

            mergeSort(arr\_merge, 0, n - 1, comparisons, swaps);

            total\_comparisons\_merge += comparisons;

            total\_swaps\_merge += swaps;

        }

        cout << "Interchange Sort\t - Comparisons: " << total\_comparisons\_interchange / num\_trials

             << ", Swaps: " << total\_swaps\_interchange / num\_trials << endl;

        cout << "Bubble Sort\t\t - Comparisons: " << total\_comparisons\_bubble / num\_trials

             << ", Swaps: " << total\_swaps\_bubble / num\_trials << endl;

        cout << "Shaker Sort\t\t - Comparisons: " << total\_comparisons\_shaker / num\_trials

             << ", Swaps: " << total\_swaps\_shaker / num\_trials << endl;

        cout << "Insertion Sort\t\t - Comparisons: " << total\_comparisons\_insertion / num\_trials

             << ", Assignments: " << total\_assignments\_insertion / num\_trials << endl;

        cout << "Binary Insertion Sort\t - Comparisons: " << total\_comparisons\_binary\_insertion / num\_trials

             << ", Assignments: " << total\_assignments\_binary\_insertion / num\_trials << endl;

        cout << "Selection Sort\t\t - Comparisons: " << total\_comparisons\_selection / num\_trials

             << ", Swaps: " << total\_swaps\_selection / num\_trials << endl;

        cout << "Shell Sort\t\t - Comparisons: " << total\_comparisons\_shell / num\_trials

             << ", Swaps: " << total\_swaps\_shell / num\_trials << endl;

        cout << "Heap Sort\t\t - Comparisons: " << total\_comparisons\_heap / num\_trials

             << ", Swaps: " << total\_swaps\_heap / num\_trials << endl;

        cout << "Quick Sort\t\t - Comparisons: " << total\_comparisons\_quick / num\_trials

             << ", Swaps: " << total\_swaps\_quick / num\_trials << endl;

        cout << "Merge Sort\t\t - Comparisons: " << total\_comparisons\_merge / num\_trials

             << ", Swaps: " << total\_swaps\_merge / num\_trials << endl;

        cout << "\n";

    }

    return 0;

}

#### Kết quả:

For n = 10:

Interchange Sort - Comparisons: 45, Swaps: 25

Bubble Sort - Comparisons: 45, Swaps: 21

Shaker Sort - Comparisons: 45, Swaps: 22

Insertion Sort - Comparisons: 33, Assignments: 33

Binary Insertion Sort - Comparisons: 22, Assignments: 35

Selection Sort - Comparisons: 45, Swaps: 7

Shell Sort - Comparisons: 32, Swaps: 10

Heap Sort - Comparisons: 63, Swaps: 26

Quick Sort - Comparisons: 25, Swaps: 17

Merge Sort - Comparisons: 22, Swaps: 34

For n = 100:

Interchange Sort - Comparisons: 4950, Swaps: 2469

Bubble Sort - Comparisons: 4950, Swaps: 2327

Shaker Sort - Comparisons: 4950, Swaps: 2569

Insertion Sort - Comparisons: 2564, Assignments: 2564

Binary Insertion Sort - Comparisons: 531, Assignments: 2638

Selection Sort - Comparisons: 4950, Swaps: 94

Shell Sort - Comparisons: 921, Swaps: 418

Heap Sort - Comparisons: 1256, Swaps: 578

Quick Sort - Comparisons: 667, Swaps: 381

Merge Sort - Comparisons: 538, Swaps: 672

For n = 200:

Interchange Sort - Comparisons: 19900, Swaps: 10002

Bubble Sort - Comparisons: 19900, Swaps: 9932

Shaker Sort - Comparisons: 19900, Swaps: 9920

Insertion Sort - Comparisons: 9981, Assignments: 9981

Binary Insertion Sort - Comparisons: 1262, Assignments: 10296

Selection Sort - Comparisons: 19900, Swaps: 194

Shell Sort - Comparisons: 2307, Swaps: 1104

Heap Sort - Comparisons: 2925, Swaps: 1362

Quick Sort - Comparisons: 1615, Swaps: 908

Merge Sort - Comparisons: 1280, Swaps: 1544

For n = 500:

Interchange Sort - Comparisons: 124750, Swaps: 61179

Bubble Sort - Comparisons: 124750, Swaps: 62059

Shaker Sort - Comparisons: 124750, Swaps: 62409

Insertion Sort - Comparisons: 62812, Assignments: 62812

Binary Insertion Sort - Comparisons: 3797, Assignments: 62333

Selection Sort - Comparisons: 124750, Swaps: 493

Shell Sort - Comparisons: 6665, Swaps: 3159

Heap Sort - Comparisons: 8566, Swaps: 4033

Quick Sort - Comparisons: 4962, Swaps: 2800

Merge Sort - Comparisons: 3848, Swaps: 4488

For n = 1000:

Interchange Sort - Comparisons: 499500, Swaps: 240424

Bubble Sort - Comparisons: 499500, Swaps: 253712

Shaker Sort - Comparisons: 499500, Swaps: 250146

Insertion Sort - Comparisons: 248634, Assignments: 248634

Binary Insertion Sort - Comparisons: 8583, Assignments: 250781

Selection Sort - Comparisons: 499500, Swaps: 991

Shell Sort - Comparisons: 15602, Swaps: 7596

Heap Sort - Comparisons: 19158, Swaps: 9079

Quick Sort - Comparisons: 10977, Swaps: 6149

Merge Sort - Comparisons: 8704, Swaps: 9976

For n = 2000:

Interchange Sort - Comparisons: 1999000, Swaps: 938252

Bubble Sort - Comparisons: 1999000, Swaps: 997734

Shaker Sort - Comparisons: 1999000, Swaps: 1000511

Insertion Sort - Comparisons: 1003584, Assignments: 1003584

Binary Insertion Sort - Comparisons: 19173, Assignments: 1007265

Selection Sort - Comparisons: 1999000, Swaps: 1992

Shell Sort - Comparisons: 37390, Swaps: 19384

Heap Sort - Comparisons: 42337, Swaps: 20168

Quick Sort - Comparisons: 24988, Swaps: 13339

Merge Sort - Comparisons: 19409, Swaps: 21952

For n = 5000:

Interchange Sort - Comparisons: 12497500, Swaps: 5362623

Bubble Sort - Comparisons: 12497500, Swaps: 6218265

Shaker Sort - Comparisons: 12497500, Swaps: 6246207

Insertion Sort - Comparisons: 6238080, Assignments: 6238080

Binary Insertion Sort - Comparisons: 54492, Assignments: 6242919

Selection Sort - Comparisons: 12497500, Swaps: 4991

Shell Sort - Comparisons: 116689, Swaps: 61684

Heap Sort - Comparisons: 119150, Swaps: 57075

Quick Sort - Comparisons: 71278, Swaps: 37595

Merge Sort - Comparisons: 55233, Swaps: 61808

For n = 10000:

Interchange Sort - Comparisons: 49995000, Swaps: 18597264

Bubble Sort - Comparisons: 49995000, Swaps: 25034520

Shaker Sort - Comparisons: 49995000, Swaps: 24990966

Insertion Sort - Comparisons: 25072409, Assignments: 25072409

Binary Insertion Sort - Comparisons: 118998, Assignments: 25019766

Selection Sort - Comparisons: 49995000, Swaps: 9989

Shell Sort - Comparisons: 266752, Swaps: 146747

Heap Sort - Comparisons: 258415, Swaps: 124207

Quick Sort - Comparisons: 154378, Swaps: 82106

Merge Sort - Comparisons: 120468, Swaps: 133616

#### Nhận xét:

1. **Interchange Sort, Bubble Sort, Shaker Sort**:
   * **Độ phức tạp**: – Không hiệu quả với mảng lớn.
   * **Comparisions** và **Swaps** tăng nhanh khi kích thước mảng tăng.
   * Không được tối ưu cho mảng lớn.
2. **Insertion Sort**:
   * **Độ phức tạp**: trong trường hợp tồi tệ, nhưng rất nhanh đối với các mảng gần sắp xếp.
   * **Comparisions** và **Assignments** tương đối ít, nhưng vẫn có số lần **Assignments** khá cao khi mảng lớn.
3. **Binary Insertion Sort**:
   * **Độ phức tạp**: , nhưng giảm số lần **Comparisions** nhờ việc sử dụng tìm kiếm nhị phân. Tuy nhiên, số lần **Assignments** không thay đổi.
4. **Selection Sort**:
   * **Độ phức tạp**: – số lần **Swaps** thấp, nhưng **Comparisions** vẫn rất nhiều.
5. **Shell Sort**:
   * **Độ phức tạp**: Không xác định chính xác, nhưng trong một số trường hợp, giúp giảm số lần **Comparisions** và **Swaps** so với các thuật toán .
6. **Heap Sort**:
   * **Độ phức tạp**: , rất hiệu quả đối với mảng lớn.
   * Số lần **Swaps** vẫn cao hơn **Quick Sort** và **Merge Sort**.
7. **Quick Sort**:
   * **Độ phức tạp**: trong trường hợp trung bình.
   * **Comparisions** và **Swaps** thấp, hiệu quả với mảng lớn, nhưng có thể gặp tình huống tồi tệ với lựa chọn pivot không hợp lý.
8. **Merge Sort**:
   * **Độ phức tạp**: rất ổn định và đáng tin cậy, nhưng yêu cầu bộ nhớ phụ.
   * **Comparisions** và **Swaps** hợp lý, nhưng số **Swaps** có thể cao hơn **Quick Sort**.

#### **Kết luận**:

* **Quick Sort** và **Merge Sort** là lựa chọn tối ưu cho mảng lớn, với **Quick Sort** thường hiệu quả hơn trong hầu hết các trường hợp.
* **Heap Sort** là lựa chọn tốt khi không cần bộ nhớ phụ, nhưng số lần hoán đổi có thể cao hơn một chút.
* Các thuật toán như **Bubble Sort**, **Shaker Sort**, **Selection Sort**, và **Insertion Sort** có độ phức tạp và không thích hợp cho mảng lớn.

### Bài 4:

#include <iostream>

#include <fstream>

#include <ctime>

#include <chrono>

#include <algorithm>

#define MAX\_SIZE 30000

void generate\_data(const std::string &filename, int num\_elements = MAX\_SIZE)

{

    std::ofstream outfile(filename);

    if (outfile.is\_open())

    {

        for (int i = 0; i < num\_elements; ++i)

        {

            outfile << rand() % 100000 + 1 << std::endl;

        }

        outfile.close();

    }

}

int read\_data(const std::string &filename, int arr[], int max\_size)

{

    std::ifstream infile(filename);

    int num\_elements = 0;

    while (infile >> arr[num\_elements] && num\_elements < max\_size)

    {

        ++num\_elements;

    }

    return num\_elements;

}

// Interchange Sort

void interchange\_sort(int arr[], int n)

{

    for (int i = 0; i < n; ++i)

    {

        for (int j = i + 1; j < n; ++j)

        {

            if (arr[i] > arr[j])

            {

                std::swap(arr[i], arr[j]);

            }

        }

    }

}

// Selection Sort

void selection\_sort(int arr[], int n)

{

    for (int i = 0; i < n; ++i)

    {

        int min\_index = i;

        for (int j = i + 1; j < n; ++j)

        {

            if (arr[j] < arr[min\_index])

            {

                min\_index = j;

            }

        }

        std::swap(arr[i], arr[min\_index]);

    }

}

// Insertion Sort

void insertion\_sort(int arr[], int n)

{

    for (int i = 1; i < n; ++i)

    {

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] > key)

        {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

// Bubble Sort

void bubble\_sort(int arr[], int n)

{

    for (int i = 0; i < n; ++i)

    {

        for (int j = 0; j < n - i - 1; ++j)

        {

            if (arr[j] > arr[j + 1])

            {

                std::swap(arr[j], arr[j + 1]);

            }

        }

    }

}

// Quick Sort

void quick\_sort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pivot = arr[high];

        int i = low - 1;

        for (int j = low; j < high; ++j)

        {

            if (arr[j] < pivot)

            {

                i++;

                std::swap(arr[i], arr[j]);

            }

        }

        std::swap(arr[i + 1], arr[high]);

        int pi = i + 1;

        quick\_sort(arr, low, pi - 1);

        quick\_sort(arr, pi + 1, high);

    }

}

// Merge Sort

void merge(int arr[], int left, int mid, int right)

{

    int n1 = mid - left + 1;

    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; ++i)

        L[i] = arr[left + i];

    for (int i = 0; i < n2; ++i)

        R[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;

    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;

    }

}

void merge\_sort(int arr[], int left, int right)

{

    if (left < right)

    {

        int mid = left + (right - left) / 2;

        merge\_sort(arr, left, mid);

        merge\_sort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

// Heap Sort

void heapify(int arr[], int n, int i)

{

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < n && arr[left] > arr[largest])

    {

        largest = left;

    }

    if (right < n && arr[right] > arr[largest])

    {

        largest = right;

    }

    if (largest != i)

    {

        std::swap(arr[i], arr[largest]);

        heapify(arr, n, largest);

    }

}

void heap\_sort(int arr[], int n)

{

    for (int i = n / 2 - 1; i >= 0; --i)

    {

        heapify(arr, n, i);

    }

    for (int i = n - 1; i > 0; --i)

    {

        std::swap(arr[0], arr[i]);

        heapify(arr, i, 0);

    }

}

double measure\_time\_interchange(int arr[], int n)

{

    auto start = std::chrono::high\_resolution\_clock::now();

    interchange\_sort(arr, n);

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double> duration = end - start;

    return duration.count();

}

double measure\_time\_selection(int arr[], int n)

{

    auto start = std::chrono::high\_resolution\_clock::now();

    selection\_sort(arr, n);

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double> duration = end - start;

    return duration.count();

}

double measure\_time\_insertion(int arr[], int n)

{

    auto start = std::chrono::high\_resolution\_clock::now();

    insertion\_sort(arr, n);

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double> duration = end - start;

    return duration.count();

}

double measure\_time\_bubble(int arr[], int n)

{

    auto start = std::chrono::high\_resolution\_clock::now();

    bubble\_sort(arr, n);

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double> duration = end - start;

    return duration.count();

}

double measure\_time\_quick(int arr[], int n)

{

    auto start = std::chrono::high\_resolution\_clock::now();

    quick\_sort(arr, 0, n - 1);

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double> duration = end - start;

    return duration.count();

}

double measure\_time\_merge(int arr[], int n)

{

    auto start = std::chrono::high\_resolution\_clock::now();

    merge\_sort(arr, 0, n - 1);

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double> duration = end - start;

    return duration.count();

}

double measure\_time\_heap(int arr[], int n)

{

    auto start = std::chrono::high\_resolution\_clock::now();

    heap\_sort(arr, n);

    auto end = std::chrono::high\_resolution\_clock::now();

    std::chrono::duration<double> duration = end - start;

    return duration.count();

}

void save\_results(const std::string &filename, const std::string &name, double time\_taken)

{

    std::ofstream outfile(filename, std::ios\_base::app); // Append mode

    if (outfile.is\_open())

    {

        outfile << name << ": " << time\_taken << " seconds\n";

        outfile.close();

    }

}

int main()

{

    const std::string input\_filename = "input\_data.txt";

    const std::string output\_filename = "output\_results.txt";

    generate\_data(input\_filename);

    int data[MAX\_SIZE];

    int n = read\_data(input\_filename, data, MAX\_SIZE);

    double time\_taken;

    // Interchange Sort

    int data\_copy[MAX\_SIZE];

    std::copy(data, data + n, data\_copy);

    time\_taken = measure\_time\_interchange(data\_copy, n);

    save\_results(output\_filename, "Interchange Sort", time\_taken);

    // Selection Sort

    std::copy(data, data + n, data\_copy);

    time\_taken = measure\_time\_selection(data\_copy, n);

    save\_results(output\_filename, "Selection Sort", time\_taken);

    // Insertion Sort

    std::copy(data, data + n, data\_copy);

    time\_taken = measure\_time\_insertion(data\_copy, n);

    save\_results(output\_filename, "Insertion Sort", time\_taken);

    // Bubble Sort

    std::copy(data, data + n, data\_copy);

    time\_taken = measure\_time\_bubble(data\_copy, n);

    save\_results(output\_filename, "Bubble Sort", time\_taken);

    // Quick Sort

    std::copy(data, data + n, data\_copy);

    time\_taken = measure\_time\_quick(data\_copy, n);

    save\_results(output\_filename, "Quick Sort", time\_taken);

    // Merge Sort

    std::copy(data, data + n, data\_copy);

    time\_taken = measure\_time\_merge(data\_copy, n);

    save\_results(output\_filename, "Merge Sort", time\_taken);

    // Heap Sort

    std::copy(data, data + n, data\_copy);

    time\_taken = measure\_time\_heap(data\_copy, n);

    save\_results(output\_filename, "Heap Sort", time\_taken);

    std::cout << "Results saved to " << output\_filename << std::endl;

    return 0;

}

## B. Bài tập vận dụng

### Bài 1:

#include <iostream>

#include <cmath>

#define MAX\_SIZE 30000

int sum\_of\_digits(int n)

{

    int sum = 0;

    n = abs(n);

    while (n > 0)

    {

        sum += n % 10;

        n /= 10;

    }

    return sum;

}

bool is\_prime(int n)

{

    if (n <= 1)

        return false;

    if (n == 2)

        return true;

    for (int i = 2; i <= sqrt(n); ++i)

    {

        if (n % i == 0)

            return false;

    }

    return true;

}

int remove\_primes(int arr[], int &n)

{

    int new\_size = 0;

    for (int i = 0; i < n; ++i)

    {

        if (!is\_prime(arr[i]))

        {

            arr[new\_size++] = arr[i];

        }

    }

    n = new\_size;

    return n;

}

void sort\_descending(int arr[], int n)

{

    for (int i = 0; i < n - 1; ++i)

    {

        for (int j = i + 1; j < n; ++j)

        {

            if (arr[i] < arr[j])

            {

                int temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }

        }

    }

}

void sort\_by\_digit\_sum(int arr[], int n)

{

    for (int i = 0; i < n - 1; ++i)

    {

        for (int j = i + 1; j < n; ++j)

        {

            if (sum\_of\_digits(arr[i]) > sum\_of\_digits(arr[j]))

            {

                int temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }

        }

    }

}

void find\_k\_largest\_elements(int arr[], int n, int k)

{

    sort\_descending(arr, n);

    std::cout << "The " << k << " largest elements are:\n";

    for (int i = 0; i < k && i < n; ++i)

    {

        std::cout << arr[i] << " ";

    }

    std::cout << std::endl;

}

int main()

{

    int n;

    std::cout << "Enter the number of elements: ";

    std::cin >> n;

    int arr[MAX\_SIZE];

    std::cout << "Enter the elements: ";

    for (int i = 0; i < n; ++i)

    {

        std::cin >> arr[i];

    }

    int k;

    std::cout << "Enter k: ";

    std::cin >> k;

    find\_k\_largest\_elements(arr, n, k);

    sort\_by\_digit\_sum(arr, n);

    std::cout << "Array sorted by the sum of digits: ";

    for (int i = 0; i < n; ++i)

    {

        std::cout << arr[i] << " ";

    }

    std::cout << std::endl;

    remove\_primes(arr, n);

    std::cout << "Array after removing prime numbers: ";

    for (int i = 0; i < n; ++i)

    {

        std::cout << arr[i] << " ";

    }

    std::cout << std::endl;

    return 0;

}

#### Kết quả

Enter the number of elements: 10

Enter the elements: 10 5 3 6 9 23 25 87 11 0

Enter k: 5

The 5 largest elements are:

87 25 23 11 10

Array sorted by the sum of digits: 0 10 11 3 23 5 6 25 9 87

Array after removing prime numbers: 0 10 6 25 9 87

### Bài 2:

#include <iostream>

struct Term

{

    float coeff;

    int degree;

};

void swap(Term &a, Term &b)

{

    Term temp = {0, 0};

    temp.degree = a.degree;

    a.degree = b.degree;

    b.degree = temp.degree;

}

void selectionSort(Term arr[], int n)

{

    for (int i = 0; i < n - 1; ++i)

    {

        int min = i;

        for (int j = i + 1; j < n; ++j)

        {

            if (arr[min].degree > arr[j].degree)

            {

                min = j;

            }

        }

        if (min != i)

            swap(arr[min], arr[i]);

    }

}

void display\_terms(Term arr[], int n)

{

    for (int i = 0; i < n; ++i)

    {

        std::cout << "Coefficient: " << arr[i].coeff << ", Degree: " << arr[i].degree << "\n";

    }

}

int main()

{

    Term terms[] = {{3.2, 5}, {1.5, 2}, {4.7, 8}, {2.1, 1}, {0.5, 7}};

    int n = sizeof(terms) / sizeof(terms[0]);

    std::cout << "Before sorting:" << "\n";

    display\_terms(terms, n);

    selectionSort(terms, n);

    std::cout << "\nAfter sorting:" << "n\n";

    display\_terms(terms, n);

    return 0;

}

#### Kết quả

Before sorting:

Coefficient: 3.2, Degree: 5

Coefficient: 1.5, Degree: 2

Coefficient: 4.7, Degree: 8

Coefficient: 2.1, Degree: 1

Coefficient: 0.5, Degree: 7

After sorting:n

Coefficient: 3.2, Degree: 1

Coefficient: 1.5, Degree: 2

Coefficient: 4.7, Degree: 5

Coefficient: 2.1, Degree: 7

Coefficient: 0.5, Degree: 8

### Bài 3:

#include <iostream>

#include <string>

struct Room

{

    int number;

    char house;

    int storageCap;

};

template <typename T>

void Swap(T &a, T &b)

{

    T temp = a;

    a = b;

    b = temp;

}

template <typename T>

void selectionSort(T arr[], int n, bool (\*cmp)(T, T))

{

    for (int i = 0; i < n - 1; ++i)

    {

        int key = i;

        for (int j = i + 1; j < n; ++j)

        {

            if (cmp(arr[key], arr[j]))

            {

                key = j;

            }

        }

        if (key != i)

        {

            Swap(arr[key], arr[i]);

        }

    }

}

bool cmpByHouseAndNumber(Room a, Room b)

{

    if (a.house == b.house)

    {

        return a.number > b.number;

    }

    return a.house > b.house;

}

bool cmpByHouseAndStorageCap(Room a, Room b)

{

    if (a.house == b.house)

    {

        return a.storageCap < b.storageCap;

    }

    return a.house > b.house;

}

bool cmpByStorageCap(Room a, Room b)

{

    return a.storageCap > b.storageCap;

}

void displayRooms(Room arr[], int n)

{

    for (int i = 0; i < n; ++i)

    {

        std::cout << "Room " << arr[i].number << " - " << arr[i].house << " - " << arr[i].storageCap << "\n";

    }

}

int main()

{

    Room rooms[] = {

        {101, 'A', 50},

        {102, 'B', 40},

        {103, 'C', 60},

        {104, 'D', 30},

        {105, 'A', 30},

        {106, 'A', 60}};

    int n = sizeof(rooms) / sizeof(rooms[0]);

    std::cout << "Original Rooms: " << std::endl;

    displayRooms(rooms, n);

    selectionSort(rooms, n, cmpByStorageCap);

    std::cout << "\nSorted by storageCap: " << std::endl;

    displayRooms(rooms, n);

    selectionSort(rooms, n, cmpByHouseAndNumber);

    std::cout << "\nSorted by House, after Number: " << std::endl;

    displayRooms(rooms, n);

    selectionSort(rooms, n, cmpByHouseAndStorageCap);

    std::cout << "\nSorted by House, after StorageCap: " << std::endl;

    displayRooms(rooms, n);

    return 0;

}

#### Kết quả:

Original Rooms:

Room 101 - A - 50

Room 102 - B - 40

Room 103 - C - 60

Room 104 - D - 30

Room 105 - A - 30

Room 106 - A - 60

Sorted by storageCap:

Room 104 - D - 30

Room 105 - A - 30

Room 102 - B - 40

Room 101 - A - 50

Room 103 - C - 60

Room 106 - A - 60

Sorted by House, after Number:

Room 101 - A - 50

Room 105 - A - 30

Room 106 - A - 60

Room 102 - B - 40

Room 103 - C - 60

Room 104 - D - 30

Sorted by House, after StorageCap:

Room 106 - A - 60

Room 101 - A - 50

Room 105 - A - 30

Room 102 - B - 40

Room 103 - C - 60

Room 104 - D – 30

### Bài 4:

#include <iostream>

#include <cmath>

bool isPrime(int n)

{

    if (n < 2)

        return false;

    int k = sqrt(n);

    for (int i = 2; i <= k; i++)

    {

        if (n % i == 0)

            return false;

    }

    return true;

}

int findTheLargestPrime(int arr[][100], int m, int n)

{

    int largest = -1;

    for (int i = 0; i < m; ++i)

    {

        for (int j = 0; j < n; ++j)

        {

            if (isPrime(arr[i][j]) && arr[i][j] > largest)

            {

                largest = arr[i][j];

            }

        }

    }

    return largest;

}

void findRowsWithPrime(int arr[][100], int m, int n)

{

    int marked[100] = {0};

    for (int i = 0; i < m; ++i)

    {

        for (int j = 0; j < n; ++j)

        {

            if (isPrime(arr[i][j]))

            {

                marked[i] = 1;

                break;

            }

        }

    }

    for (int i = 0; i < m; ++i)

    {

        if (marked[i] == 1)

        {

            for (int j = 0; j < n; ++j)

            {

                std::cout << arr[i][j] << " ";

            }

            std::cout << "\n";

        }

    }

}

void findRowsWithOnlyPrime(int arr[][100], int m, int n)

{

    int marked[100] = {0};

    for (int i = 0; i < m; ++i)

    {

        for (int j = 0; j < n; ++j)

        {

            if (!isPrime(arr[i][j]))

            {

                marked[i] = 1;

                break;

            }

        }

    }

    for (int i = 0; i < m; ++i)

    {

        if (marked[i] == 0)

        {

            for (int j = 0; j < n; ++j)

            {

                std::cout << arr[i][j] << " ";

            }

            std::cout << "\n";

        }

    }

}

int main()

{

    int arr[4][100] = {

        {2, 3, 4},

        {5, 6, 7},

        {8, 9, 10},

        {11, 13, 17}};

    int largestPrime = findTheLargestPrime(arr, 4, 3);

    std::cout << "The largest prime in the matrix: " << largestPrime << "\n";

    std::cout << "Rows with prime number: \n";

    findRowsWithPrime(arr, 4, 3);

    std::cout << "Rows with only prime number: \n";

    findRowsWithOnlyPrime(arr, 4, 3);

    return 0;

}

#### Kết quả:

The largest prime in the matrix: 17

Rows with prime number:

2 3 4

5 6 7

11 13 17

Rows with only prime number:

11 13 17

### Bài 5:

#include <iostream>

#include <cmath>

using namespace std;

bool isPrime(int n)

{

    if (n < 2)

        return false;

    int k = sqrt(n);

    for (int i = 2; i <= k; i++)

    {

        if (n % i == 0)

            return false;

    }

    return true;

}

int sumOfRow(int arr[], int n)

{

    int sum = 0;

    for (int i = 0; i < n; ++i)

    {

        sum += arr[i];

    }

    return sum;

}

void findLargestSumofRow(int arr[][100], int m, int n)

{

    int largest = -1;

    int index = -1;

    for (int i = 0; i < m; ++i)

    {

        int sum = sumOfRow(arr[i], n);

        if (sum > largest)

        {

            largest = sum;

            index = i;

        }

    }

    std::cout << "Row " << index + 1 << " has the largest sum of " << largest << std::endl;

}

void sortBySumOfRow(int arr[][100], int m, int n)

{

    for (int i = 0; i < m - 1; ++i)

    {

        int maxIndex = i;

        for (int j = i + 1; j < m; ++j)

        {

            if (sumOfRow(arr[j], n) > sumOfRow(arr[maxIndex], n))

            {

                maxIndex = j;

            }

        }

        if (maxIndex != i)

        {

            for (int k = 0; k < n; ++k)

            {

                swap(arr[i][k], arr[maxIndex][k]);

            }

        }

    }

    std::cout << "Matrix after sorting rows by sum (in descending order):\n";

    for (int i = 0; i < m; ++i)

    {

        for (int j = 0; j < n; ++j)

        {

            std::cout << arr[i][j] << " ";

        }

        std::cout << "\n";

    }

}

int main()

{

    int arr[5][100] = {

        {1, 2, 3, 4},

        {5, 6, 7, 8},

        {9, 10, 11, 12},

        {13, 14, 15, 16},

        {17, 18, 19, 20}};

    findLargestSumofRow(arr, 5, 4);

    sortBySumOfRow(arr, 5, 4);

    return 0;

}

#### Kết quả:

Row 5 has the largest sum of 74

Matrix after sorting rows by sum (in descending order):

17 18 19 20

13 14 15 16

9 10 11 12

5 6 7 8

1 2 3 4

### Bài 6:

#include <iostream>

using namespace std;

void selectionSortEvens(int arr[], int n)

{

    for (int i = 0; i < n - 1; ++i)

    {

        if (arr[i] != 0 && arr[i] % 2 == 0)

        {

            for (int j = i + 1; j < n; ++j)

            {

                if (arr[j] != 0 && arr[j] % 2 == 0 && arr[i] > arr[j])

                {

                    swap(arr[i], arr[j]);

                }

            }

        }

    }

}

void selectionSortOdds(int arr[], int n)

{

    for (int i = 0; i < n - 1; ++i)

    {

        if (arr[i] != 0 && arr[i] % 2 != 0)

        {

            for (int j = i + 1; j < n; ++j)

            {

                if (arr[j] != 0 && arr[j] % 2 != 0 && arr[i] < arr[j])

                {

                    swap(arr[i], arr[j]);

                }

            }

        }

    }

}

void sortArray(int arr[], int n)

{

    selectionSortEvens(arr, n);

    selectionSortOdds(arr, n);

}

int main()

{

    int arr[] = {10, 3, 0, 5, 8, 6, 0, 7, 4};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array:\n";

    for (int i = 0; i < n; ++i)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

    sortArray(arr, n);

    cout << "Array after sorting evens in ascending order and odds in descending order:\n";

    for (int i = 0; i < n; ++i)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}

#### Kết quả:

Original array:

10 3 0 5 8 6 0 7 4

Array after sorting evens in ascending order and odds in descending order:

4 7 0 5 6 8 0 3 10

### Bài 7:

#include <iostream>

using namespace std;

void sortEvenOdd(int arr[], int n)

{

    int left = 0;

    int right = n - 1;

    while (left < right)

    {

        while (left < n && arr[left] % 2 == 0)

        {

            left++;

        }

        while (right >= 0 && arr[right] % 2 != 0)

        {

            right--;

        }

        if (left < right)

        {

            swap(arr[left], arr[right]);

        }

    }

}

int main()

{

    int arr[] = {3, 1, 2, 4, 5, 6};

    int n = sizeof(arr) / sizeof(arr[0]);

    sortEvenOdd(arr, n);

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}

#### Kết quả:

6 4 2 1 5 3

### Bài 9:

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <cstring>

struct Student

{

    char id[6];

    char lastName[21];

    char firstName[41];

    char dob[11];

    char gender[7];

    double gpa;

    void setString(char \*dest, const std::string &src, size\_t maxLen)

    {

        strncpy(dest, src.c\_str(), maxLen - 1);

        dest[maxLen - 1] = '\0';

    }

    Student(const std::string &i, const std::string &ln, const std::string &fn,

            const std::string &d, const std::string &g, double gp) : gpa(gp)

    {

        setString(id, i, 6);

        setString(lastName, ln, 21);

        setString(firstName, fn, 41);

        setString(dob, d, 11);

        setString(gender, g, 7);

    }

    Student() {}

};

struct Index

{

    char id[6];

    long offset;

};

void inputStudentList(std::vector<Student> &students)

{

    students = {

        {"SV001", "Dang", "Thanh Dai", "2000-01-01", "Male", 9.5},

        {"SV002", "Ngo", "Gia Bao", "2000-02-15", "Male", 7.9},

        {"SV003", "Ho", "Huu Anh Tuan", "2000-03-10", "Male", 9.1},

        {"SV004", "Nguyen", "Thanh Vinh", "2000-04-12", "Male", 8.2},

        {"SV005", "Nguyen", "Minh Thuan", "2000-05-25", "Male", 7.8},

        {"SV006", "Nguyen", "Gia An", "2000-06-18", "Male", 9.3},

        {"SV007", "Le", "Thanh Thuy", "2000-07-30", "Female", 8.0},

        {"SV008", "Nguyen", "Lan Anh", "2000-08-20", "Female", 8.7},

        {"SV009", "Ho", "Trung Hieu", "2000-09-10", "Male", 7.5},

        {"SV010", "Nguyen", "Minh Hao", "2000-10-05", "Male", 8.3}};

}

void outputStudent(const Student &student)

{

    std::cout << "ID: " << student.id << "\n"

              << "Name: " << student.lastName << " " << student.firstName << "\n"

              << "Date of birth: " << student.dob << "\n"

              << "Gender: " << student.gender << "\n"

              << "GPA: " << student.gpa << "\n\n";

}

void saveStudentListToFile(const std::vector<Student> &students)

{

    std::ofstream file("STUDENTS.DAT", std::ios::binary);

    if (!file)

    {

        std::cerr << "Cannot open file STUDENTS.DAT!\n";

        return;

    }

    for (const auto &student : students)

    {

        file.write(reinterpret\_cast<const char \*>(&student), sizeof(Student));

    }

    file.close();

}

void sortByID(std::vector<Student> &students)

{

    int n = students.size();

    for (int i = 0; i < n - 1; i++)

    {

        for (int j = 0; j < n - i - 1; j++)

        {

            if (strcmp(students[j].id, students[j + 1].id) > 0)

            {

                Student temp = students[j];

                students[j] = students[j + 1];

                students[j + 1] = temp;

            }

        }

    }

}

void sortByName(std::vector<Student> &students)

{

    int n = students.size();

    for (int i = 0; i < n - 1; i++)

    {

        for (int j = 0; j < n - i - 1; j++)

        {

            int cmp = strcmp(students[j].firstName, students[j + 1].firstName);

            if (cmp > 0 || (cmp == 0 && strcmp(students[j].lastName, students[j + 1].lastName) > 0))

            {

                Student temp = students[j];

                students[j] = students[j + 1];

                students[j + 1] = temp;

            }

        }

    }

}

void sortByGPA(std::vector<Student> &students)

{

    int n = students.size();

    for (int i = 0; i < n - 1; i++)

    {

        for (int j = 0; j < n - i - 1; j++)

        {

            if (students[j].gpa < students[j + 1].gpa)

            {

                Student temp = students[j];

                students[j] = students[j + 1];

                students[j + 1] = temp;

            }

        }

    }

}

void createIndexFile(const std::vector<Student> &students, const std::string &filename,

                     bool (\*cmp)(const Student &, const Student &))

{

    std::vector<Index> indices;

    std::fstream dataFile("STUDENTS.DAT", std::ios::binary | std::ios::in);

    if (!dataFile)

    {

        std::cerr << "Cannot open file STUDENTS.DAT!\n";

        return;

    }

    long offset = 0;

    for (const auto &student : students)

    {

        Index idx;

        strcpy(idx.id, student.id);

        idx.offset = offset;

        indices.push\_back(idx);

        offset += sizeof(Student);

    }

    int n = indices.size();

    for (int i = 0; i < n - 1; i++)

    {

        for (int j = 0; j < n - i - 1; j++)

        {

            Student sa, sb;

            dataFile.seekg(indices[j].offset);

            dataFile.read(reinterpret\_cast<char \*>(&sa), sizeof(Student));

            dataFile.seekg(indices[j + 1].offset);

            dataFile.read(reinterpret\_cast<char \*>(&sb), sizeof(Student));

            if (cmp(sa, sb))

            {

                Index temp = indices[j];

                indices[j] = indices[j + 1];

                indices[j + 1] = temp;

            }

        }

    }

    std::ofstream idxFile(filename, std::ios::binary);

    if (!idxFile)

    {

        std::cerr << "Cannot open file " << filename << "!\n";

        return;

    }

    idxFile.write(reinterpret\_cast<const char \*>(indices.data()), indices.size() \* sizeof(Index));

    idxFile.close();

    dataFile.close();

}

void readFromIndexAndPrint(const std::string &idxFilename)

{

    std::ifstream idxFile(idxFilename, std::ios::binary);

    std::fstream dataFile("STUDENTS.DAT", std::ios::binary | std::ios::in);

    if (!idxFile || !dataFile)

    {

        std::cerr << "Cannot open file!\n";

        return;

    }

    Index idx;

    while (idxFile.read(reinterpret\_cast<char \*>(&idx), sizeof(Index)))

    {

        Student student;

        dataFile.seekg(idx.offset);

        dataFile.read(reinterpret\_cast<char \*>(&student), sizeof(Student));

        outputStudent(student);

    }

    idxFile.close();

    dataFile.close();

}

int main()

{

    std::vector<Student> students;

    inputStudentList(students);

    saveStudentListToFile(students);

    sortByID(students);

    std::cout << "List sorted by ID:\n";

    for (const auto &student : students)

        outputStudent(student);

    createIndexFile(students, "ID\_INDEX.IDX",

                    [](const Student &a, const Student &b)

                    { return strcmp(a.id, b.id) > 0; });

    createIndexFile(students, "NAME\_INDEX.IDX",

                    [](const Student &a, const Student &b)

                    {

                        int cmp = strcmp(a.firstName, b.firstName);

                        if (cmp != 0)

                            return cmp > 0;

                        return strcmp(a.lastName, b.lastName) > 0;

                    });

    createIndexFile(students, "GPA\_INDEX.IDX",

                    [](const Student &a, const Student &b)

                    { return a.gpa < b.gpa; });

    std::cout << "\nList by ID index:\n";

    readFromIndexAndPrint("ID\_INDEX.IDX");

    std::cout << "\nList by Name index:\n";

    readFromIndexAndPrint("NAME\_INDEX.IDX");

    std::cout << "\nList by GPA index:\n";

    readFromIndexAndPrint("GPA\_INDEX.IDX");

    return 0;

}

#### Kết quả:

List sorted by ID:

ID: SV001

Name: Dang Thanh Dai

Date of birth: 2000-01-01

Gender: Male

GPA: 9.5

ID: SV002

Name: Ngo Gia Bao

Date of birth: 2000-02-15

Gender: Male

GPA: 7.9

ID: SV003

Name: Ho Huu Anh Tuan

Date of birth: 2000-03-10

Gender: Male

GPA: 9.1

ID: SV004

Name: Nguyen Thanh Vinh

Date of birth: 2000-04-12

Gender: Male

GPA: 8.2

ID: SV005

Name: Nguyen Minh Thuan

Date of birth: 2000-05-25

Gender: Male

GPA: 7.8

ID: SV006

Name: Nguyen Gia An

Date of birth: 2000-06-18

Gender: Male

GPA: 9.3

ID: SV007

Name: Le Thanh Thuy

Date of birth: 2000-07-30

Gender: Female

GPA: 8

ID: SV008

Name: Nguyen Lan Anh

Date of birth: 2000-08-20

Gender: Female

GPA: 8.7

ID: SV009

Name: Ho Trung Hieu

Date of birth: 2000-09-10

Gender: Male

GPA: 7.5

ID: SV010

Name: Nguyen Minh Hao

Date of birth: 2000-10-05

Gender: Male

GPA: 8.3

List by ID index:

ID: SV001

Name: Dang Thanh Dai

Date of birth: 2000-01-01

Gender: Male

GPA: 9.5

ID: SV002

Name: Ngo Gia Bao

Date of birth: 2000-02-15

Gender: Male

GPA: 7.9

ID: SV003

Name: Ho Huu Anh Tuan

Date of birth: 2000-03-10

Gender: Male

GPA: 9.1

ID: SV004

Name: Nguyen Thanh Vinh

Date of birth: 2000-04-12

Gender: Male

GPA: 8.2

ID: SV005

Name: Nguyen Minh Thuan

Date of birth: 2000-05-25

Gender: Male

GPA: 7.8

ID: SV006

Name: Nguyen Gia An

Date of birth: 2000-06-18

Gender: Male

GPA: 9.3

ID: SV007

Name: Le Thanh Thuy

Date of birth: 2000-07-30

Gender: Female

GPA: 8

ID: SV008

Name: Nguyen Lan Anh

Date of birth: 2000-08-20

Gender: Female

GPA: 8.7

ID: SV009

Name: Ho Trung Hieu

Date of birth: 2000-09-10

Gender: Male

GPA: 7.5

ID: SV010

Name: Nguyen Minh Hao

Date of birth: 2000-10-05

Gender: Male

GPA: 8.3

List by Name index:

ID: SV006

Name: Nguyen Gia An

Date of birth: 2000-06-18

Gender: Male

GPA: 9.3

ID: SV002

Name: Ngo Gia Bao

Date of birth: 2000-02-15

Gender: Male

GPA: 7.9

ID: SV003

Name: Ho Huu Anh Tuan

Date of birth: 2000-03-10

Gender: Male

GPA: 9.1

ID: SV008

Name: Nguyen Lan Anh

Date of birth: 2000-08-20

Gender: Female

GPA: 8.7

ID: SV010

Name: Nguyen Minh Hao

Date of birth: 2000-10-05

Gender: Male

GPA: 8.3

ID: SV005

Name: Nguyen Minh Thuan

Date of birth: 2000-05-25

Gender: Male

GPA: 7.8

ID: SV001

Name: Dang Thanh Dai

Date of birth: 2000-01-01

Gender: Male

GPA: 9.5

ID: SV007

Name: Le Thanh Thuy

Date of birth: 2000-07-30

Gender: Female

GPA: 8

ID: SV004

Name: Nguyen Thanh Vinh

Date of birth: 2000-04-12

Gender: Male

GPA: 8.2

ID: SV009

Name: Ho Trung Hieu

Date of birth: 2000-09-10

Gender: Male

GPA: 7.5

List by GPA index:

ID: SV001

Name: Dang Thanh Dai

Date of birth: 2000-01-01

Gender: Male

GPA: 9.5

ID: SV006

Name: Nguyen Gia An

Date of birth: 2000-06-18

Gender: Male

GPA: 9.3

ID: SV003

Name: Ho Huu Anh Tuan

Date of birth: 2000-03-10

Gender: Male

GPA: 9.1

ID: SV008

Name: Nguyen Lan Anh

Date of birth: 2000-08-20

Gender: Female

GPA: 8.7

ID: SV010

Name: Nguyen Minh Hao

Date of birth: 2000-10-05

Gender: Male

GPA: 8.3

ID: SV004

Name: Nguyen Thanh Vinh

Date of birth: 2000-04-12

Gender: Male

GPA: 8.2

ID: SV007

Name: Le Thanh Thuy

Date of birth: 2000-07-30

Gender: Female

GPA: 8

ID: SV002

Name: Ngo Gia Bao

Date of birth: 2000-02-15

Gender: Male

GPA: 7.9

ID: SV005

Name: Nguyen Minh Thuan

Date of birth: 2000-05-25

Gender: Male

GPA: 7.8

ID: SV009

Name: Ho Trung Hieu

Date of birth: 2000-09-10

Gender: Male

GPA: 7.5