**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**



BÁO CÁO ĐỒ ÁN GIỮA KÌ

**Môn: Cấu trúc dữ liệu và giải thuật**

**Họ và tên: Âu Dương Gia Tuấn**

**MSSV: 1712867**

1. Giới thiệu chung

Về việc các *thuật toán sắp xếp* (*Sorting Algorithm*), ta có rất nhiều loại thuật toán khác nhau, ví dụ như: selection sort, insertion sort, interchange sort, binary interchange sort, radix sort, merge sort, heap sort, flash sort, counting sort, shell sort, bubble sort, shaker sort, quick sort, bucket sort, cocktail sort,….

Mỗi thuật toán trên đều có ưu và nhược điểm khác nhau, tùy thuộc vào từng bộ dữ liệu mà ta chọn mộ thuật toán thích hợp nhất để cài đặt.

Nhưng trong bài báo cáo này ta chỉ tìm hiểu về 10 thuật toán khá phổ biến hiện nay, đó là: selection sort, interchange sort, insertion sort, binary insertion sort, shaker sort, quick sort, heap sort, shell sort, merge sort, radix sort.

1. Tìm hiểu về các thuật toán

# Selection sort

## Ý tưởng

Ý tưởng của thuật toán Selection sort chia phần mảng ban đầu thành hai phần: phần mảng đã được sắp xếp và phần mảng chưa được sắp xếp.

Sau đó tiến hành sắp xếp mảng bằng cách liên tục tìm ra *phần tử nhỏ nhất (ví dụ sắp cho xếp tăng dần)* từ phần mảng chưa được sắp xếp và hoán đổi vị trí đó với vị trí đầu. Tiến trình này sẽ dừng lại khi toàn bộ phần tử của mảng chưa được sắp xếp đều được chuyển sang phần mảng đã được sắp xếp.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 3 | 15 | 1 |

Ví dụ:

**Mảng ban đầu:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 3 | 15 | 1 |

Vị trí đầu tiên chính là số 10 và sau khi duyệt mảng thì ta có số 1 là giá trị nhỏ nhất:

***i*=0**

Ta tiến hành hoán đổi số 10 và số 1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 4 | 7 | 3 | 15 | 10 |

Lúc này mảng đã được chia thành hai phần: phần đã sắp xếp và phần chưa sắp xếp

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 4 | 7 | 3 | 15 | 10 |

Tiến trình trên sẽ áp dụng cho phần còn lại của mảng.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 4 | 7 | 3 | 15 | 10 |

***i*=1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 4 | 15 | 10 |

***i*=2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 15 | 10 |

***i*=3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 15 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

***i*=4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

**Mảng đã sắp xếp:**

## Thuật toán

|  |
| --- |
| void hoanVi(int &a, int &b) |
| {  int temp;  temp = a;  a = b;  b = temp;  } |

|  |
| --- |
| void selectionSort(int\* arr, int soLuongPhanTu) |
| {  int min;  for (int i = 0; i < soLuongPhanTu - 1; i++)  {  min = i;  for (int j = i + 1; j < soLuongPhanTu; j++)  {  if (arr[min] > arr[j])  {  min = j;  }  }  if (i != min)  {  hoanVi(arr[i], arr[min]);  }  }  } |

## Đánh giá

Độ phức tạp của thuật toàn là O(

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Thuật toán dễ cài đặt.  Sắp xếp nhanh với mảng ít phần tử. | Tốn nhiều bộ nhớ.  Đối với mảng nhiều phần tử, thuật toán sắp xếp lâu. |

# Interchange sort



## Ý tưởng

Ý tưởng của thuật toán Interchange sort là bắt cặp tất cả phần tử trong mảng và đổi chổ cho nhau nếu hai phần tử đó thỏa điều kiện sắp xếp.

Thuật toán này sẽ dùng hai vòng lặp. Vòng lặp ngoài sẽ chạy từ *vị trí đầu tiên* đến vị trí thứ *n-1* của mảng. Vòng lặp trong sẽ chạy từ *vị trí* *ta đang xét + 1* đến vị trí *n* của mảng. Mỗi lần ta sẽ so sánh hai giá trị trong vòng lặp, nếu chúng thỏa điều kiện, sẽ tiến hành hoán đổi vị trí.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 3 | 15 | 1 |

**Ví dụ:**

**Mảng ban đầu:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 3 | 15 | 1 |

Ta sẽ tiến hành sắp xếp mảng trên tăng dần.

***i*=0**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 10 | 7 | 3 | 15 | 1 |

Vì số 10 và 4 thỏa điều kiện nên ta sẽ hoán đổi vị trí hai số này.

Ta sẽ tiến hành quá trinh này cho đến khi chạy hết vòng lặp.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 10 | 7 | 3 | 15 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 10 | 7 | 3 | 15 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 10 | 7 | 4 | 15 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 10 | 7 | 4 | 15 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 10 | 7 | 4 | 15 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 7 | 4 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 7 | 4 | 15 | 3 |

***i*=1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 7 | 10 | 4 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 7 | 4 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 4 | 7 | 10 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 4 | 7 | 10 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 4 | 7 | 10 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 10 | 15 | 4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 10 | 15 | 4 |

***i*=2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 10 | 15 | 4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 10 | 15 | 4 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 10 | 15 | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 10 | 15 | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 10 | 15 | 7 |

***i*=3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 15 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 15 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

***i*=4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

**Mảng đã sắp xếp:**

## Thuật toán

|  |
| --- |
| void interchangeSort(int\* arr, int soLuongPhanTu) |
| {  for (int i = 0; i < soLuongPhanTu - 1; i++)  {  for (int j = i + 1; j < soLuongPhanTu; j++)  {  if (arr[i] > arr[j])  {  hoanVi(arr[i], arr[j]);  }  }  }  } |

## Đánh giá

Độ phức tạp của thuật toàn là O(.

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Thuật toán dễ cài đặt.  Sắp xếp nhanh với mảng ít phần tử. | Tốn nhiều bộ nhớ.  Đối với mảng nhiều phần tử, thuật toán sắp xếp lâu. |

# Insertion sort

## Ý tưởng

Ý tưởng của thuật toán Insertion sort giống như việc chơi bài, khi trên tay bạn đang cầm một dãy bài đã theo thứ tự và người chơi phát cho bạn thêm một quân bài, bạn sẽ so sánh quân bài mới đó với từng quân bài trên tay bạn cho đến khi bạn tìm được vị trí thích hợp để chèn vào.

Với thuật toán Insertion sort, ta sẽ bắt đầu vòng lặp từ vị trí thứ 1 đến hết mảng, trong vòng lặp ta sẽ tìm ra phần tử cần chèn rồi di chuyển phần tử đó đến vị trí thích hợp rồi chèn vào vị trí đó.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 3 | 15 | 1 |

**Ví dụ:**

**Mảng ban đầu:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 3 | 15 | 1 |

***i*=1**

Ta sẽ bắt đầu vòng lặp từ vị trí số 4, ta thấy số 4 không theo thứ tự với vị trí đứng trước nó nên ta sẽ di chuyển số 4 và chèn vào trước số 10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 10 | 7 | 3 | 15 | 1 |

Cứ như vậy thực hiện hết vòng lặp

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 10 | 7 | 3 | 15 | 1 |

***i*=2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 7 | 10 | 3 | 15 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 7 | 10 | 3 | 15 | 1 |

**i=3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 4 | 7 | 10 | 15 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 4 | 7 | 10 | 15 | 1 |

***i*=4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 4 | 7 | 10 | 15 | 1 |

***i*=5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

**Mảng sau sắp xếp:**

## Thuật toán

|  |
| --- |
| void insertionSort(int\* arr, int soLuongPhanTu) |
| {  int x, j;  for (int i = 1; i < soLuongPhanTu; i++)  {  x = arr[i];  j = i - 1;  while ((j >= 0) && (arr[j] > x))  {  arr[j + 1] = arr[j];  j--;  }  arr[j + 1] = x;  }  } |

## Đánh giá

Độ phức tạp của thuật toán là O()

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Thuật toán dễ cài đặt.  Sắp xếp nhanh với mảng ít phần tử hoặc mảng đã được sắp xếp một phần. | Đối với mảng nhiều phần tử, thuật toán sắp xếp khá lâu.  Tốn nhiều bộ nhớ. |

# Binary Insertion sort

## Ý tưởng

Ý tưởng của thuật toán Binary Insertion sort giống như thuật toán Insertion sort nhưng trong Binary Insertion sort dùng thuật toán tìm kiếm Binary để tìm kiếm vị trí cần chèn.

## Thuật toán

|  |
| --- |
| int timKiemNhiPhan(int\* arr, int x, int left, int right) |
| {  if (right <= left)  {  return (x > arr[left]) ? (left + 1) : left;  }  int mid = (left + right) / 2;  if (arr[mid] == x)  {  return mid + 1;  }  else if (arr[mid] > x)  {  return timKiemNhiPhan(arr, x, left, mid - 1);  }  else  {  return timKiemNhiPhan(arr, x, mid + 1, right);  }  } |

|  |
| --- |
| void binaryInsertionSort(int\* arr, int soLuongPhanTu) |
| {  int x, j, y;  for (int i = 1; i < soLuongPhanTu; i++)  {  x = arr[i];  j = i - 1;  y = timKiemNhiPhan(arr, x, 0, j);  while (j >= y)  {  arr[j + 1] = arr[j];  j--;  }  arr[j + 1] = x;  }  } |

## Đánh giá

Độ phức tạp của thuật toán: O(log(n))

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Sắp xếp nhanh với mảng ít phần tử hoặc mảng đã được sắp xếp một phần. | Tốn nhiều bộ nhớ.  Đối với mảng nhiều phần tử, thuật toán sắp xếp khá lâu. |

# Shaker sort

## Ý tưởng

Ý tưởng thuật toán Shaker sort là xét ba vòng lặp,vòng lặp thứ nhất ta xét trong khi left<=right, vòng lặp thứ hai bắt đầu từ cuối dãy, vòng lặp thứ ba bắt đầu từ đầu dãy. Trong vòng lặp thứ hai, ta xét hai phần tử đứng liền nhau nếu chúng thỏa điều kiện, thì hoán đổi vị trí, kết thúc vòng lập thứ hai thì phần tử nhỏ nhất sẽ đứng đầu mảng (nếu như ta sắp xếp mảng tăng dần). Trong vòng lặp thứ hai, ta cũng xét hai phần tử đứng liền nhau nếu chúng thỏa điều kiện, thì hoán đổi vị trí, kết thúc vòng lập thứ ba thì phần tử lớn nhất sẽ đứng cuối mảng (nếu như ta sắp xếp mảng tăng dần).

Ví dụ:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 3 | 1 |

**Mảng ban đầu:**

*Left=0, right 5*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 3 | 1 |

***While(left<=right)***

**Vòng lặp thứ hai:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 3 | 1 | 3 |

***i*=5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 1 | 3 |

***i*=4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 1 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 1 | 15 | 3 |

***i*=3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 1 | 7 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 1 | 7 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 1 | 4 | 7 | 15 | 3 |

***i*=2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 1 | 4 | 7 | 15 | 3 |

***i*=1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 15 | 3 |

Kết thúc vòng lặp thứ hai ta có giá trị nhỏ nhất đứng đầu mảng:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 15 | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 15 | 3 |

Vòng lặp thứ ba:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 15 | 3 |

***i*=1**

***i*=2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 15 | 3 |

***i*=3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 15 | 3 |

***i*=4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 15 | 3 |

***i*=5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 3 | 15 |

Kết thúc vòng lặp thứ ba ta có gia trị lớn nhất đứng cuối mảng:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 4 | 7 | 3 | 15 |

Tiếp tục thực hiện quá trình trên cho đến khi kết thúc vòng lặp while(left<=right), ta sẽ được mãng sắp xếp hoàn chỉnh

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

## Thuật toán

|  |
| --- |
| void shakerSort(int\* arr, int soLuongPhanTu) |
| {  int left = 1, right = soLuongPhanTu - 1, i, k;  do  {  for (i = right; i >= left; i--)  {  if (arr[i - 1] > arr[i])  {  hoanVi(arr[i], arr[i - 1]);  }  k = i;  }  left = k + 1;  for (i = left; i <= right; i++)  {  if (arr[i - 1] > arr[i])  {  hoanVi(arr[i], arr[i - 1]);  }  k = i;  }  right = k - 1;  } while (left <= right);  } |

## Đánh giá

Độ phức tạp của thuật toàn là O(

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Thuật toán dễ dàng cài đặt  Sắp xếp nhanh với mảng ít phần tử. | Tốn nhiều bộ nhớ.  Đối với mảng nhiều phần tử, thuật toán sắp xếp rất lâu. |

# Shell sort

## Ý tưởng

Shell sort có ý tưởng khá giống với thuật toán Insertion sort. Tuy nhiên trong thuật toán Insertion sort có lúc ta cần di chuyển một phần tử đi một khoảng xa để chèn vào vị trí thích hợp thì đối với thuật toán Shell sort ta sắp xếp hai giá trị có *khoảng cách xa nhau* (*gap*), và sau đó giảm giá trị gap giữa hai giá trị cần được so sánh dần về 1.

Ví dụ:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 3 | 1 |

**Mảng ban đầu:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 3 | 1 |

Ta tiến hành sắp xếp tăng dần:

***gap*=3, *i*=3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 3 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 4 | 1 |

*i***=4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 3 | 7 | 15 | 4 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 3 | 7 | 15 | 4 | 1 |

***i*=5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 3 | 7 | 15 | 4 | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 3 | 1 | 15 | 4 | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 3 | 1 | 15 | 4 | 7 |

***gap*=1, *i*=1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 10 | 1 | 15 | 4 | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 10 | 1 | 15 | 4 | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 10 | 1 | 15 | 4 | 7 |

***i*=2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 10 | 10 | 15 | 4 | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 3 | 10 | 15 | 4 | 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 10 | 15 | 4 | 7 |

Tiến hành tiếp tục cho đến khi gap = 0 thì thuật toán kết thúc, lúc đó ta sẽ có mảng sắp xếp hoàn chỉnh

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

## Thuật toán

|  |
| --- |
| void shellSort(int\* arr, int soLuongPhanTu) |
| {  for (int gap = soLuongPhanTu / 2 ; gap > 0 ; gap /= 2)  {  for (int i = gap; i < soLuongPhanTu; 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;  }  }  } |

## Đánh giá

Độ phức tạp của thuật toàn là O(

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Sắp xếp nhanh. | Thuật toán tương đối khó cài đặt.  Tốn nhiều bộ nhớ. |

# Quik sort

## Ý tưởng

Ý tưởng của thuật toán Quick sort là chọn ra một điểm làm mốc (pivot), sau đó chia mảng ra thành hai phần, phần mảng trước pivot và phần mảng sau pivot, lần lượt so sánh các phần tử với pivot, nếu phần tử có giá trị lớn hơn pivot thì sẽ đưa vào phần sau, ngược lại nếu phần tử có giá trị nhỏ hơn pivot thì sẽ đưa vào phần mảng trước pivot. Sau đó, ta tiền hành sắp xếp tương tự như vậy cho phần mảng trước pivot và phần mảng sau pivot, cho đến khi mảng có độ dài bằng 1.

Ví dụ:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 15 | 7 | 4 | 3 | 1 |

Mảng ban đầu

Pivot=n/2=3;

Ta tiến hành sắp xếp mảng tăng dần

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 15 | 7 | 4 | 3 | 1 |

***i*=0, *j*=5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 15 | 7 | 4 | 3 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 15 | 7 | 4 | 3 | 10 |

***i*=1, *j*=4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 4 | 15 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 7 | 4 | 15 | 10 |

***i*=2, *j*=3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 15 | 10 |

***i*=3, *j*=2** => vòng lặp kết thúc

Tiếp tục thực hiện sắp xếp đối vởi dãy trướ pivot và sau pivot cho đến khi dãy có độ dài bằng 1 thì ta sẽ thu được dãy được sắp xếp hoàn chỉnh.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

## Thuật toán

|  |
| --- |
| void quickSort(int\* arr, int left, int right) |
| {  if (left < right)  {  int key = arr[(right + left) / 2];  int i = left;  int j = right;  while (i <= j)  {  while (arr[i]<key)  {  i++;  }  while (arr[j]>key)  {  j--;  }  if (i <= j)  {  hoanVi(arr[i], arr[j]);  i++;  j--;  }  }  if (left < j)  quickSort(arr, left, j);  if (right > i)  quickSort(arr, i, right);  }  } |

## Đánh giá

Độ phức tạp của thuật toàn là O(

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Sắp xếp nhanh. | Thuật toán tương đối khó cài đặt.  Tốn nhiều bộ nhớ. |

# 2.8. Merge sort

## 2.8.1. Ý tưởng

Ý tưởng của thuật toán Merge sort là chia đôi mảng lớn thành các mảng con nhỏ hơn, và ta tiếp tục chia đôi các mảng con đó cho đến khi mảng con nhỏ nhất chỉ có một phần tử. Sau đó, chúng ta tiến hành vừa so sánh các mảng con vừa sắp xếp và ghép hai mảng con đó thành một mảng, cứ tiếp tục như vậy cho đến khi còn lại một mảng duy nhất, đó chính là mảng đã sắp xếp xong.

Ví dụ:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 3 | 1 |

**Mảng cho trước:**

|  |  |  |
| --- | --- | --- |
| 10 | 4 | 7 |

|  |  |  |
| --- | --- | --- |
| 15 | 3 | 1 |

Ta tiến hành chia đôi mảng:

|  |  |
| --- | --- |
| 15 | 3 |

|  |
| --- |
| 1 |

|  |  |
| --- | --- |
| 10 | 4 |

|  |
| --- |
| 7 |

|  |
| --- |
| 15 |

|  |
| --- |
| 1 |

|  |
| --- |
| 3 |

|  |
| --- |
| 10 |

|  |
| --- |
| 4 |

|  |
| --- |
| 7 |

|  |  |
| --- | --- |
| 4 | 10 |

|  |
| --- |
| 7 |

|  |  |
| --- | --- |
| 3 | 15 |

|  |
| --- |
| 1 |

**Sắp xếp:**

|  |  |  |
| --- | --- | --- |
| 4 | 7 | 10 |

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 15 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

## 2.8.2. Thuật toán

|  |
| --- |
| void merge(int \*arr, int left, int mid, int right) |
| {  int n1 = mid - left + 1;  int n2 = right - mid;  int i, j, k;  int \*L = new int[n1];  int \*R = new int[n2];  for (i = 0; i < n1; i++)  {  L[i] = arr[left + i];  }  for (j = 0; j < n2; j++)  {  R[j] = arr[mid + 1 + j];  }  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++;  }  delete[] L;  delete[] R;  } |
|  |

|  |
| --- |
| void mergeSort(int \*arr, int left, int right) |
| {  if (left < right)  {  int mid = (left + right) / 2;  mergeSort(arr, left, mid);  mergeSort(arr, mid + 1, right);  merge(arr, left, mid, right);  }  } |

## 2.8.3. Đánh giá

Độ phức tạp thuật toán O(nlog(n))

Thuật toán sắp xếp có độ phức tạp trung bình.

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Sắp xếp nhanh. | Thuật toán tương đối khó cài đặt.  Không nhận diện được mảng đã sắp xếp.  Tốn nhiều bộ nhớ. |

# 2.9. Heap sort

## 2.9.1. Ý tưởng

Ý tưởng của thuật toán Heap sort là chia phần tử thành 2 phần, phần đã được sắp xếp và phần chưa được sắp xếp. Trong mảng chưa được sắp xếp ta sẽ lần lượt chọn ra cá phần tử lớn nhất bằng cấu trúc dữ liệu heap và đưa nó vào phần mảng đã được sắp xếp.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 4 | 7 | 15 | 3 | 1 |

Ví dụ:

**Mảng cho trước:**

|  |
| --- |
| 10 |

Tìm phần tử lớn nhất bằng cấu trúc heap:

|  |
| --- |
| 4 |

|  |
| --- |
| 7 |

|  |
| --- |
| 15 |

|  |
| --- |
| 3 |

|  |
| --- |
| 1 |

|  |
| --- |
| 15 |

|  |
| --- |
| 10 |

|  |
| --- |
| 7 |

|  |
| --- |
| 4 |

|  |
| --- |
| 3 |

|  |
| --- |
| 1 |

Đổi chổ vị trí đầu mảng với vị trí cuối mảng và sau đó loại phần tử lớn nhất khỏi cấu trúc heap:

|  |
| --- |
| 1 |

|  |
| --- |
| 10 |

|  |
| --- |
| 7 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 10 | 7 | 4 | 3 | 15 |

|  |
| --- |
| 4 |

|  |
| --- |
| 3 |

Tiếp tục thực hiện quá trình trên:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 10 |  |  | | --- | | 4 |  |  | | --- | | 1 |  |  | | --- | | 7 |  |  | | --- | | 3 | | |  | | --- | | 3 |  |  | | --- | | 4 |  |  | | --- | | 1 |  |  | | --- | | 7 |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 3 | 4 | 7 | 1 | 10 | 15 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 7 |  |  | | --- | | 4 |  |  | | --- | | 1 |  |  | | --- | | 3 | | |  | | --- | | 1 |  |  | | --- | | 4 |  |  | | --- | | 3 |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 1 | 4 | 3 | 7 | 10 | 15 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 4 |  |  | | --- | | 1 |  |  | | --- | | 3 | | |  | | --- | | 3 |  |  | | --- | | 1 |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 3 | 1 | 4 | 7 | 10 | 15 | |
| |  | | --- | | 3 |  |  | | --- | | 1 | | |  | | --- | | 1 |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 1 | 3 | 4 | 7 | 10 | 15 | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 7 | 10 | 15 |

Ta có mảng sắp xếp:

## 2.9.2. Thuật toán

|  |
| --- |
| void heapify(int \*arr, int soLuongPhanTu, int i) |
| {  int m = i;  int left = 2 \* i + 1;  int right = 2 \* i + 2;  if (left < soLuongPhanTu && arr[left] > arr[m])  {  m = left;  }  if (right < soLuongPhanTu && arr[right] > arr[m])  {  m = right;  }  if (m != i)  {  hoanVi(arr[i], arr[m]);  heapify(arr, soLuongPhanTu, m);  }  } |

|  |
| --- |
| void heapSort(int \*arr, int soLuongPhanTu) |
| {  for (int i = soLuongPhanTu / 2 - 1; i >= 0; i--)  {  heapify(arr, soLuongPhanTu, i);  }  for (int i = soLuongPhanTu - 1; i >= 0; i--)  {  hoanVi(arr[0], arr[i]);  heapify(arr, i, 0);  }  } |

## 2.9.3. Đánh giá

Độ phức tạp của thuật toán O(nlog(n))

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Sắp xếp nhanh. | Thuật toán tương đối khó cài đặt.  Tốn nhiều bộ nhớ. |

# 2.10. Radix sort

## 2.10.1. Ý tưởng

Ý tưởng của thuật toán Radix sort là sắp xếp dựa trên cơ số. Ta lần lượt sắp xếp các số trong mảng theo cơ số hàng đơn vị, chục, trăm, tùy theo số lớn nhất trong mảng mà ta sẽ sắp xếp đến cơ số đó.

Ví dụ:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101 | 5 | 85 | 60 | 55 |

Mảng cho trước:

Ta sắp xếp theo hàng đơn vị:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 55 |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |
| 85 |  |  |  |  |  | 55 |  |  |  |  |
| 5 |  |  |  |  |  | 85 |  |  |  |  |
| 101 | 60 | 101 |  |  |  | 5 |  |  |  |  |
| Mảng | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Sắp xếp theo hàng chục:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 55 |  |  |  |  |  |  |  |  |  |  |
| 85 |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |
| 101 | 5 |  |  |  |  |  |  |  |  |  |
| 60 | 101 |  |  |  |  | 55 | 60 |  | 85 |  |
| Mảng | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Sắp xếp theo hàng trăm:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 85 |  |  |  |  |  |  |  |  |  |  |
| 60 | 85 |  |  |  |  |  |  |  |  |  |
| 55 | 60 |  |  |  |  |  |  |  |  |  |
| 5 | 55 |  |  |  |  |  |  |  |  |  |
| 101 | 5 | 101 |  |  |  |  |  |  |  |  |
| Mảng | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 55 | 60 | 85 | 101 |

Mảng đã được sắp xếp:

## 2.10.2. Thuật toán

|  |
| --- |
| int timPhanTuLonNhat(int \*arr, int soLuongPhanTu) |
| {  int max = arr[0];  for (int i = 0; i < soLuongPhanTu; i++)  {  if (arr[i] > max)  {  max = arr[i];  }  }  return max;  } |

|  |
| --- |
| void demSort(int \*arr, int soLuongPhanTu, int m) |
| {  int \*output = new int[soLuongPhanTu];  int i;  int dem[10] = { 0 };  for (i = 0; i < soLuongPhanTu; i++)  {  dem[(arr[i] / m) % 10]++;  }  for (i = 1; i < 10; i++)  {  dem[i] += dem[i - 1];  }  for (i = soLuongPhanTu - 1; i >= 0; i--)  {  output[dem[(arr[i] / m) % 10] - 1] = arr[i];  dem[(arr[i] / m) % 10]--;  }  for (i = 0; i < soLuongPhanTu; i++)  {  arr[i] = output[i];  }  delete[] output;  } |

|  |
| --- |
| void radixSort(int \*arr, int soLuongPhanTu) |
| {  int max = timPhanTuLonNhat(arr, soLuongPhanTu);  for (int m = 1; max / m > 0; m \*= 10)  {  demSort(arr, soLuongPhanTu, m);  }  } |

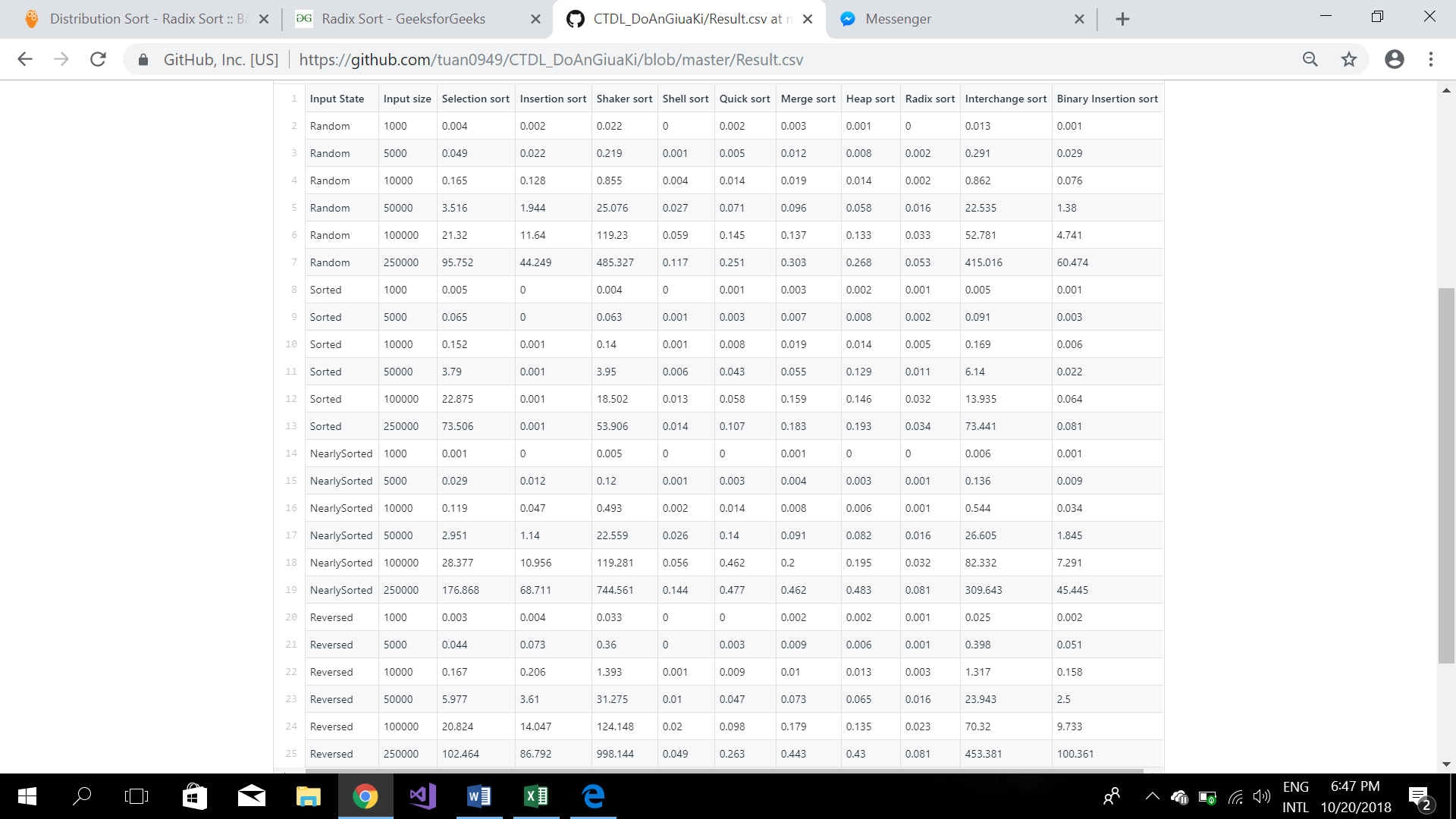
## 2.10.3 Đánh giá

Độ phức tạp của thuật toán O(n)

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| Sắp xếp nhanh.  Tốn ít bộ nhớ | Thuật toán tương đối khó cài đặt.  Chỉ sắp xếp được số nguyên |

1. Kết quả thí nghiệm

# Bảng tổng kết thời gian chạy



## Phân tích dữ liệu

## Trạng thái Random

**Nhận xét:**

Với bộ dữ liệu 1000, 5000, 10000 phần tử thì 10 thuật toán đều cho ra kết quả khá nhanh, khoảng chưa đến 1 giây.

Tuy nhiên đối với bộ dữ liệu 50000 phần tử thì đã có sự khác biệt, Shaker sort là thuật toán chạy lâu nhất với khoảng 25 giây, thuật toán chạy lâu thứ hai là Interchange sort với khoảng 23 giây, các thuật toán có tốc độ chạy trung bình là Insertion sort, Binary Insertion sort và Selection sort với khoảng thời gian chạy khá nhanh khoảng từ 1 đến 4 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Radix sort là thuật toán chạy nhanh nhất với 0.016 giây.

Với bộ dữ liệu 100000 phần tử, Shaker sort là thuật toán chạy lâu nhất với khoảng 119 giây, thuật toán chạy nhanh thứ hai là Interchange sort vơi khoảng 53 giây, Insertion sort, Binary Insertion sort và Selection sort là các thuật toán có tốc độ chạy trung bình với khoảng thời gian chạy từ 4 đến 22 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Radix sort là thuật toán chạy nhanh nhất với 0.033 giây.

Với bộ dữ liệu 250000 phần tử, Shaker sort là thuật toán chạy lâu nhất với khoảng 485 giây, thuật toán chạy nhanh thứ hai là Interchange sort vơi khoảng 415 giây, Insertion sort, Binary Insertion sort và Selection sort là các thuật toán có tốc độ chạy trung bình với khoảng thời gian chạy từ 44 đến 96 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Radix sort là thuật toán chạy nhanh nhất với 0.053 giây.

Nhìn chung với trạng thái Random thì Shaker sort và Interchange sort là hai thuật toán chạy lâu, Insertion sort, Binary Insertion sort và Selection sort là các thuật toán trung bình, các thuật toán còn lại khá tối ưu với thời gian chạy nhanh.

## Trạng thái Sorted

**Nhận xét:**

Với bộ dữ liệu 1000, 5000, 10000 phần tử thì 10 thuật toán đều cho ra kết quả khá nhanh, khoảng chưa đến 1 giây.

Tuy nhiên đối với bộ dữ liệu 50000 phần tử thì đã có sự khác biệt, Interchange sort là thuật toán chạy lâu nhất với khoảng 6.14 giây, các thuật toán có tốc độ chạy trung bình là Shker sort và Selection sort với khoảng thời gian chạy khá nhanh khoảng từ 3.8 đến 3.95 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Insertion sort là thuật toán chạy nhanh nhất với 0.001 giây.

Với bộ dữ liệu 100000 phần tử, Selection sort là thuật toán chạy lâu nhất với khoảng 22,8 giây, thuật toán chạy lâu thứ hai là Shaker sort vơi khoảng 18,5 giây, Interchange sort là các thuật toán có tốc độ chạy trung bình với khoảng thời gian chạy từ 14 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Insertion sort là thuật toán chạy nhanh nhất với 0.001 giây.

Với bộ dữ liệu 250000 phần tử, Selection sort là thuật toán chạy lâu nhất với khoảng 73,5 giây, thuật toán chạy lâu thứ hai là Interchange sort vơi khoảng 73,4 giây, thuật toán chạy lâu thứ 3 là Shaker sort với khoảng 53,9 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Insertion sort là thuật toán chạy nhanh nhất với 0.001 giây.

Nhìn chung với trạng thái Sorted thì Selection sort, Shaker sort, Interchange sort là các thuật toán chạy lâu, các thuật toán còn lại thì thời gian chạy nhanh chưa đến 1 giây.

## Trạng thái Nearly Sorted

**Nhận xét:**

Với bộ dữ liệu 1000, 5000, 10000 phần tử thì 10 thuật toán đều cho ra kết quả khá nhanh, khoảng chưa đến 1 giây.

Tuy nhiên đối với bộ dữ liệu 50000 phần tử thì đã có sự khác biệt, Interchange sort là thuật toán chạy lâu nhất với khoảng 26.6 giây, thuật toán có thời gian chạy lâu thứ hai là Shaker sort với 22,56 giây, các thuật toán có tốc độ chạy trung bình là Selection sort, Insertion sort và Binary Insertion sort với khoảng thời gian chạy khá nhanh khoảng từ 1,1 đến 3 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Radix sort là thuật toán chạy nhanh nhất với 0.016 giây.

Với bộ dữ liệu 100000 phần tử, Shaker sort là thuật toán chạy lâu nhất với khoảng 119,2 giây, thuật toán có thời gian chạy lâu thứ hai là Interchange sort với 82,3 giây, các thuật toán có tốc độ chạy trung bình là Selection sort, Insertion sort và Binary Insertion sort với khoảng thời gian chạy khá nhanh khoảng từ 7,3 đến 28 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Radix sort là thuật toán chạy nhanh nhất với 0.032 giây.

Với bộ dữ liệu 250000 phần tử, Shaker sort là thuật toán chạy lâu nhất với khoảng 744,5 giây, thuật toán có thời gian chạy lâu thứ hai là Interchange sort với 309,6 giây và thuật toán có thời gian chạy lâu thư ba là Selection sort với khoảng 176,9 giây, các thuật toán có tốc độ chạy trung bình là Insertion sort và Binary Insertion sort với khoảng thời gian chạy khá nhanh khoảng từ 45 đến 69 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Radix sort là thuật toán chạy nhanh nhất với 0.081 giây.

Nhìn chung với trạng thái Nearly Sorted thì Shaker sort, Interchange sort là các thuật toán chạy lâu, riêng thuật toán Selection sort thì tùy vào kích thước bộ dữ liệu mà ảnh hưởng đến thời gian chạy của chương trình, Insertion sort và Binary Insertion sort là hai thuật toán có tốc độ chạy trung bình, các thuật toán còn lại thì thời gian chạy nhanh chưa đến 1 giây.

## Tình trạng Reversed

**Nhận xét:**

Với bộ dữ liệu 1000, 5000 phần tử thì 10 thuật toán đều cho ra kết quả khá nhanh, khoảng chưa đến 1 giây.

Với bộ dữ liệu 10000 phần tử thì Shaker sort là thuật có thời gian sắp xếp chậm nhất với 1,393 giây, Interchange sort là thuật toán có thời gian sắp xếp chậm thứ hai với 1,317 giây, các thuật toán còn lại có thời gian sắp xếp nhanh với chưa đến 1 giây.

Với bộ dữ liệu 50000, Shaker sort là thuật toán chạy lâu nhất với khoảng 31,275 giây, thuật toán có thời gian chạy lâu thứ hai là Interchange sort với 23,943 giây, các thuật toán có tốc độ chạy trung bình là Selection sort, Insertion sort và Binary Insertion sort với khoảng thời gian chạy khá nhanh khoảng từ 2,5 đến 5,9 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Shell sort là thuật toán chạy nhanh nhất với 0.01 giây.

Với bộ dữ liệu 100000 phần tử, Shaker sort là thuật toán chạy lâu nhất với khoảng 124,148 giây, thuật toán có thời gian chạy lâu thứ hai là Interchange sort với 70,32 giây, các thuật toán có tốc độ chạy trung bình là Selection sort, Insertion sort và Binary Insertion sort với khoảng thời gian chạy khá nhanh khoảng từ 9,7 đến 20,8 giây, các thuật toán còn lại có tốc độ chạy nhanh, chưa đến 1 giây trong đó Shell sort là thuật toán chạy nhanh nhất với 0.02 giây.

Với bộ dữ liệu 250000 phần tử, Shaker sort là thuật toán chạy lâu nhất với khoảng 998,114 giây, thuật toán có thời gian chạy lâu thứ hai là Interchange sort với 453,381 giây, các thuật toán có tốc độ chạy trung bình là Selection sort, Insertion sort và Binary Insertion sort với khoảng thời gian chạy khá nhanh khoảng từ khoảng 86,8 đến khoảng 102,5 giây, các thuật toán còn lại có tốc độ chạy nhanh chưa đến 1 giây trong đó Shell sort là thuật toán chạy nhanh nhất với 0.049 giây.

Nhìn chung với trạng thái Reversed thì Shaker sort, Interchange sort là các thuật toán chạy lâu, các thuât toán Selection sort, Insertion sort và Binary Insertion sort là ba thuật toán có tốc độ chạy trung bình, các thuật toán còn lại thì thời gian chạy nhanh chưa đến 1 giây.

# Nhận xét chung

Nhìn chung với mọi kiểu tình trạng thì các thuật toán Shell sort, Quick sort, Merge sort, Heap sort, Radix sort là những thuật toán khá tối ưu, khi có thể sắp xếp nhanh với thời gian chưa đến 1 giây với mọi kích thước bộ dữ liệu từ 1000 đến 250000, tuy nhiên đối với các thuật toán này khá khó về mặt lập trình, và các thuật toán Shell sort, Quick sort, Merge sort và Heap sort có nhược điểm là khá tốn bộ nhớ khi sắp xếp vì số lượng phép so sánh và phép gán khá nhiều. Trong khi đó thuật toán Radix sort lại có ưu điểm là ít tốn bộ nhớ nhưng với thuật toán này có một nhước điểm là chỉ có thể sắp xếp được mảng là số nguyên.

Còn đối với thuật toán Selection sort, Insertion sort, Shaker sort, Interchange sort, Binary Insertion sort lại có ưu điểm là thuật toán khá dễ về mặt lập trình, và đối với mảng có kích thước phần tử nhỏ thì các thuật toán này sắp xếp khá nhanh. Còn đối với mảng có kích thước phần tử lớn thì các thuật toán này sắp xếp khá chậm. Tuy nhiên đối với hai thuật toán Insertion sort và Binary Insertion sort khi mảng nhập vào đã sắp xếp sẳn thì thời gian sắp xếp của hai thuật toán này lại khá nhanh. Ngoài ra 5 thuật toán trên lại có một nhược điểm khác chính là tốn rất nhiều bộ nhớ để sắp xếp vì số lượng phép so sánh và phép gán rất nhiều.