# Data Structures and Algorithms

21KDL1-21280115-21280075
ĐH KHOA HỌC TỰ NHIÊN TP HCM

**THỰC HÀNH CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

# BÁO CÁO TUẦN 3-5

## SEARCH - SORT

Tài liệu này ghi nhận lại nội dung các thuật toán search và sort bao gồm:

A. Search

Linear search, Binary search.

B. Sort

Selection Sort, Insertion Sort, Binary Insertion Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, Flash Sort.

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

# Nội dung

# I.Thông tin chung

Các thành viên tham dự:

| STT | MSSV | Họ và tên | Email |
|---|---|---|---|
| 1 | 21280115 | Trần Đức Trung | 21280115@student.hcmus.edu.vn |
| 2 | 21280075 | Nguyễn Hoàng Thông | 21280075@student.hcmus.edu.vn |

Mục tiêu:

1. Hiểu được ý tưởng các thuật toán search và sort
2. Cài đặt thành công và vẽ trực quan so sánh các thuật toán

Thời gian bắt đầu: .....3-10-2022..... Thời gian kết thúc: .....24-10-2022.....

# II.Nội dung

# A. SEARCH

## 1. Linear Search

- **Ý tưởng chung:** Duyệt từ phần tử đầu tiên đến phần từ cuối cùng đến khi tìm được phần tử có giá trị bằng giá trị cần tìm.

- **Mã giả:**

```
Begin (a[], n, x)
For i = 0 to n do
    If (a[i] == x)
        Return i
    Endif
Endfor
Return -1
End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Phần tử cần tìm ở đầu mảng | Phần tử cần tìm ở giữa mảng | Phần tử cần tìm ở cuối hoặc không nằm trong mảng |
| Độ phức tạp | O(n) | O(n) | O(n) |

- **VD đơn giản với mảng 5 phần tử:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1** : Cần tìm phần tử có giá trị x= 24

**B2**: Duyệt từ i = 0 đến i = 3 và a[3] = 24

**B3**: Kết quả trả ra: i = 3

## 2. Binary Search

- **Ý tưởng chung:** Dùng trong mảng đã được sắp xếp bằng cách chia đôi mảng cần tìm kiếm, nếu phần tử cần tìm nhỏ hơn phần tử ở giữa thì thu hẹp phạm vi tìm kiếm vào bên trái mảng, ngược lại thì thu hẹp phạm vi tìm kiếm ở bên phải mảng.

- **Mã giả:**

```
Begin (a[], left, right, key)
If (right >= left)
    mid = (left + right)/2
    If (a[mid] == key)
        Return mid
    Endif
    If (a[mid] > key)
        Return binarySearch(a, left, mid – 1, key)
    Endif
    Return binarySearch(a, mid + 1, right, key)
Endif
Return -1
End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Phần tử cần tìm là phần tử ở giữa | Phần tử cần tìm nằm bên trái hoặc phải so với phần tử ở giữa | Phần tử cần tìm không nằm trong mảng |
| Độ phức tạp | O(1) | O(log n) | O(log n) |

- **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

| 1 | 2 | 5 | 17 | 24 |
|---|---|---|---|---|

**B1**: Cần tìm phần tử x = 2

left = 0, right = 4

mid = (left + right)/2 = 2

**B2**: Vì a[mid] > x (5 > 2)

| 1 | 2 |
|---|---|

Cập nhật left = không đổi, right = mid-1

mid lúc này bằng (left + right)/2 => bằng 0

Vì a[mid] < x (1<2)

| 2 |
|---|

Cập nhật left = mid+1, right không đổi

mid lúc này bằng (left + right)/2 => bằng 1

a[mid] bằng với x bằng 2, return mid

**B3**: Kết quả trả ra : 1

# B. SORT

## 1. Selection Sort

- **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Mảng được chia thành 2 mảng con là mảng đã và chưa được sắp xếp. Selection Sort tìm phần tử nhỏ nhất trong mảng chưa được sắp xếp và đổi chỗ với phần tử ở đầu mảng chưa sắp xếp. Tức là mỗi vòng lặp, phần tử nhỏ nhất của mảng chưa sắp xếp sẽ được chuyển đến mảng đã sắp xếp.

- **Mã giả:**

```
Function swap(a, b):
        temp = a
        a = b
        b = temp

Begin (a[], n)
i, j
For i = 0 to n-1 do
      min = i
      For j =  i+1 to n do
            If (a[j] < a[min])
                min = j
            Endif
      Endfor
      if ( min != i )
            swap(a[min], a[j])
      Endif
Endfor
End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp | Mảng lộn xộn | Mảng bị đảo ngược |
| Độ phức tạp | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| Stable | Không stable | | |

- **VD đơn giản với mảng 5 phần tử:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1**: i = 1. Lặp lần thứ 1

Tìm phần tử nhỏ nhất trong a[0....4] : min = a[2] = 1. Đổi chỗ với phần tử đầu tiên

| | | | | |
|---|---|---|---|---|
| 1 | 5 | 2 | 17 | 24 |

**B2**: i = 2. Lặp lần thứ 2

Tìm phần tử nhỏ nhất trong a[1...4] : min = a[2] = 2. Đổi chỗ với phần tử đầu tiên của a[1...4]

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 24 | 17 |

**B3**: i = 3. Lặp lần thứ 3

Tìm phần tử nhỏ nhất trong a[2..4] : min = a[2] = 5. Giữ nguyên vị trí

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 24 | 17 |

**B4**: i = 4. Lặp lần thứ 4

Tìm phần tử nhỏ nhất trong a[3...4]: min = a[4] = 17. Đổi chỗ với phần tử đầu tiên của a[3...4]

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

Mảng sau khi sắp xếp :

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

## 2. Insertion Sort

- **Ý tưởng chung**: Giả sử sắp xếp mảng tăng dần. Bắt đầu duyệt từ phần tử thứ 2 trở đi, so sánh phần tử hiện tại với phần tử trước nó. Nếu phần tử hiện tại lớn hơn thì đổi chỗ 2 phần tử với nhau, sau đó tiếp tục so sánh và đổi chỗ như vậy ở các vòng lặp tiếp theo sao cho đảm bảo tính tăng dần của mảng cho đến khi hoàn thành sắp xếp.

- **Mã giả:**

```
Begin (a[] ,n)
i, j, current
For i = 1 to n do
      current = a[j]
      j = i – 1
      while (j >=0 and a[j] > current) do
            a[j+1] = a[j]
            j = j -1
       endwhile
       a[j +1] = current
Endfor
End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|

| Trường hợp | Mảng đã được sắp xếp, chỉ có vòng lặp bên ngoài chạy n lần | Mảng có thứ tự lộn xộn | Mảng có thứ tự bị đảo ngược |
|---|---|---|---|
| Độ phức tạp | O(n) | $O(n^2)$ | $O(n^2)$ |
| Stable | Stable | | |

- **VD đơn giản với mảng 5 phần tử:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1**: i = 1. Lặp lần thứ 1

Vì 5 lớn hơn 2 nên giữ nguyên vị trí của 5.

| 2 | 5 | 1 | 17 | 24 |
|---|---|---|---|---|

**B2**: i = 2. Lặp lần thứ 2

Vì 1 nhỏ hơn cả 5 và 2 nên 1 sẽ di chuyển về đầu dãy, các phần tử 2 và 5 sẽ di chuyển sang phải 1 vị trí so với hiện tại.

| 1 | 2 | 5 | 24 | 17 |
|---|---|---|---|---|

**B3**: i = 3. Lặp lần thứ 3

Vì 24 lớn hơn tất cả phần tử từ a[0...2] nên 24 vẫn giữ nguyên vị trí

| 1 | 2 | 5 | 24 | 17 |
|---|---|---|---|---|

**B4**: i = 4. Lặp lần thứ 4

Vì 17 lớn hơn 1,2,5 nhưng nhỏ hơn 24 nên 17 sẽ di chuyển về vị trí sau 5, phần tử 24 sẽ di chuyển về sau 1 vị trí

| 1 | 2 | 5 | 17 | 24 |
|---|---|---|---|---|

Mảng sau khi sắp xếp :

| 1 | 2 | 5 | 17 | 24 |
|---|---|---|---|---|

## 3. Binary Insertion Sort

- **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Mảng được chia thành 2 mảng con đã và chưa được sắp xếp. Duyệt từ phần tử thứ 2 đến phần tử cuối cùng, phần tử hiện tại là key. Tiếp đó dùng Binary Search vào mảng đã được sắp xếp để tìm vị trí "pos" có giá trị lớn hơn key. Dịch chuyển toàn bộ phần tử từ "pos" về 1 sang phải để tạo vị trí trống cho key.

- **Mã giả:**

```
Function binarySearch(a, key, left, right)
        If ( right <= left)
                If (key> a[left]) return (left + 1)
                Else return left
        mid =(left + right) / 2
        If (a[mid] == key) return mid+1
        If (key > a[mid])
                Return binarySearch(a ,key ,mid+1 ,right)
        Return binarySearch(a ,key ,left ,mid - 1)
```

```
Begin (a[] ,n)
Idx, key
For i =1 to n-1 do
```

```
        j = i – 1
        key = a[i]
        idx = binarySearch(a, 0, j, key)
        while ( j >= idx) do
            a[j +1] = a[j]
            j = j – 1
        endwhile
        a[j+1] = key
    Endfor
    End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp, chỉ có vòng lặp bên ngoài chạy n lần | Mảng có thứ tự lộn xộn | Mảng có thứ tự bị đảo ngược |
| Độ phức tạp | O(n) | $O(n^2)$ | $O(n^2)$ |
| Stable | Stable | | |

- **VD đơn giản với mảng 5 phần tử:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1:** Giả sử phần tử đầu tiên đã được sắp xếp. Key = a[1] = 5

Dùng binary search : 2 < 5 nên giữ nguyên vị trí

| | | | | |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B2**: Key = a[2] = 1

Vì 2 > 1 nên dịch chuyển 2 và 5 sang phải một vị trí.

a[0] = 1.

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 24 | 17 |

**B3**: Key = a[3] = 24

Dùng binary search : 24 > 5 nên giữ nguyên vị trí

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 24 | 17 |

**B4**: Key = a[4] = 17

Dùng binary search tìm được phần tử lớn hơn 17 trong a[0...3] là 24

Dịch chuyển 24 sang phải một vị trí, lúc này a[3] = 17

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

Mảng kết quả:

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

## 4. Bubble Sort

- **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Ở mỗi vòng lặp, so sánh 2 phần tử liền kề và đổi chỗ nếu chúng xếp sai thứ tự tăng dần. Lặp lại cho đến khi hoàn thành sắp xếp mảng.

- **Mã giả:**

```
Begin (a[], n)
i, j
For i = 0 to n-1 do
    For j = 0 to n – i – 1 do
        If (a[j] > a[j+1])
```

```
                swap(a[j], a[j+1])
            Endif
        Endfor
    Endfor
    End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự lộn xộn | Mảng có thứ tự bị đảo ngược |
| Độ phức tạp | O(n) | $O(n^2)$ | $O(n^2)$ |
| Stable | Stable | | |

- **VD đơn giản với mảng 5 phần tử:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1**: i = 1. Lặp lần thứ 1

So sánh a[0] và a[1]. Giữ nguyên vị trí hai phần tử do 5 > 2

| | | | | |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

So sánh a[1] và a[2]. Đổi chỗ 2 phần tử do 5 > 1

| | | | | |
|---|---|---|---|---|
| 2 | 1 | 5 | 24 | 17 |

So sánh a[2] và a[3]. Giữ nguyên vị trí 2 phần tử do 5 < 24

| | | | | |
|---|---|---|---|---|
| 2 | 1 | 5 | 24 | 17 |

So sánh a[3] và a[4]. Đổi chỗ hai phần tử do 24 > 17

| | | | | |
|---|---|---|---|---|
| 2 | 1 | 5 | 17 | 24 |

**B2** : i = 2. Lặp lần thứ 2

So sánh a[0] và a[1]. Đổi chỗ 2 phần tử do 2 > 1

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

So sánh a[1] và a[2]. Giữ nguyên vị trí do 2 < 5

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

So sánh a[2] và a[3]. Giữ nguyên vị trí do 5 < 17

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

So sánh a[3] và a[4]. Giữ nguyên vị trí do 17 < 24

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

Mảng sau khi sắp xếp :

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

## 5. Shaker Sort

- **Ý tưởng chung:**

Shaker Sort là một cải tiến của Bubble Sort.

Sau khi đưa phần tử nhỏ nhất về đầu mảng sẽ đưa phần tử lớn nhất về cuối dãy. Do đưa các phần tử về đúng vị trí ở cả hai đầu nên Shaker Sort sẽ giúp cải thiện thời gian sắp xếp dãy số do giảm được độ lớn của mảng đang xét ở lần so sánh kế tiếp.

- **Mã giả:**

Begin (arr[], l, r):

```
Left=l, right=r, k=0
While( left <= right)
        for i=left to right-1 do
                if (arr[i] > arr[i+1])
                        swap(arr[i] , arr[i+1])
                        k=i
                end if
        end for
        right=k
        for i=right to leff+1 do
                if (arr[i] < arr[i-1])
                        swap(arr[i] , arr[i-1])
                        k=i
                end if
        end for
        left=k
end while
end
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(n) | O(n^2) | O(n^2) |
| Stable | Stable | | |

- **Ví dụ:**

**Tạo 2 tham số right=4 và left=0. Nếu right<left thì dừng**

| | | | | |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1: Duyệt từ left sang right nếu a[trái] > a[phải] thì swap. Đưa được số lớn nhất về cuối mảng. Cập nhật right=3**

| | | | | |
|---|---|---|---|---|
| 2 | 1 | 5 | 24 | 17 |

| | | | | |
|---|---|---|---|---|
| 2 | 1 | 5 | 17 | 24 |

**B2: Duyệt từ right về left nếu a[phải] < a[trái] thì swap. Đưa số bé nhất về đầu mảng. Cập nhật left=1**

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

**B3: tiếp tục lặp lại bước 1 và 2 tới khi right < left và dừng**

## 6. Shell Sort

- **Ý tưởng chung:** Chọn ra 1 giá trị gap, đánh số từ 1 đến gap cho các phần tử trong mảng ví dụ gap là 3 thì các phần tử được đánh số là 1 2 3 1 2 3 1 2 3 ... Từ đó coi các phần tử được đánh cùng 1 số là 1 mảng con và thực hiện insertion sort trên các mảng con đó.Sau khi thực hiện xong bước trên, giá trị của gap sẽ được thu nhỏ dần về 1 và kết thúc
- **Mã giả:**

```
Begin (arr[] , l , r)
For ( gap=(r-l+1)/2 ;gap >0 ; gap /= 2 ) do
        For i=gap to i==r do
                Temp=arr[i]
                For ( j=i; j>=gap and arr[i-gap] > temp; j-=gap ) do
                        arr[j] = arr[j-gap]
                end for
                arr[j] = temp
        end for
end for
end
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |

| Độ phức tạp | O(nlog(n)) | O(nlog(n)) | O(n^2) |
|---|---|---|---|
| Stable | Stable | | |

- **Ví dụ:**

| | a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|---|
| | 2 | 5 | 1 | 24 | 17 |
| **Gap =2, i=2** | | | | | |
| | **1** | 5 | **2** | 24 | 17 |
| **Gap=2 , i=3** | | | | | |
| | 1 | **5** | 2 | **24** | 17 |
| **Gap =2 , i=4** | | | | | |
| | **1** | 5 | **2** | 24 | **17** |
| **Gap=1,i=1** | | | | | |
| | 1 | 5 | 2 | 24 | 17 |
| **Gap=1,i=2** | | | | | |
| | 1 | **2** | **5** | 24 | 17 |
| **Gap=1,i=3** | | | | | |
| | 1 | 2 | **5** | **24** | 17 |
| **Gap=1,i=4** | | | | | |
| | 1 | 2 | 5 | **17** | **24** |

## 7. Heap Sort

- **Ý tưởng chung:** Tạo ra 1 cây max tree, và mô phỏng nó trên mảng. Sau đó ta sẽ đổi chỗ node max và node min ( tương ứng với vị trí tại mảng), như vậy giá trị lớn nhất đã được sắp xếp, liên tục lặp lại đến khi mảng được sắp xếp hoàn thiện
- **Mã giả:**

**// Hàm kiểm tra và thực hiện tính chất của max heap tại từng node**

```
Function heapify(a[], size, i)
        nodeFather = i              // Lưu lại vị trí node father
        left=2*i+1                  // lưu node left của node father
        right=2*i+2                 // lưu node right
```

```
        if (left < size and a[left] > a[nodeFather] )
                nodeFather=left
        end if
        if (right < size and a[right] > a[nodeFather] )
                nodeFather=right
        end if
        if ( nodeFather != i )              // nếu node father ko phải lớn nhất, swap với node con đó
                swap(a[i], a[nodeFather] )
            // kiểm tra node con sau khi đổi có giữ tính chất của heap ko
                heapify(a, size, nodeFather )
        end if
```

```
Begin(a[] , size)
// Vòng lặp chạy qua từng nodeFather trong tree, bắt đầu tại node father cuối
For i=size/2 -1 to i==0 do:
        heapify(a, size, i)
end for
for i=size-1 to i==0 do:
        swap( a[0] ,a[i] )          // Đưa node max về cuối mảng
        heapify(a , i , 0 )         // Tạo max heap mới ( không tính node max ở cuối mảng )
end for
end
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(nlog(n)) | O(nlog(n)) | O(nlog(n)) |
| Stable | Không stable | | |

- **Ví dụ:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**Ta minh họa bằng tree được**

| | | | | |
|---|---|---|---|---|
| | | 2 | | |
| | 5 | | 1 | |
| 24 | 17 | | | |

| | | | | |
|---|---|---|---|---|
| | | 2 | | |
| | 24 | | 1 | |
| 5 | 17 | | | |

| | | | | |
|---|---|---|---|---|
| | | 24 | | |
| | 2 | | 1 | |
| 5 | 17 | | | |

| | | | | |
|---|---|---|---|---|
| | | 24 | | |
| | 17 | | 1 | |
| 5 | 2 | | | |

| | | | | |
|---|---|---|---|---|
| | | 2 | | |
| | 17 | | 1 | |
| 5 | 24 | | | |

| | | | | |
|---|---|---|---|---|
| | | 17 | | |
| | 2 | | 1 | |
| 5 | 24 | | | |

| | | | | |
|---|---|---|---|---|
| | | 17 | | |
| | 5 | | 1 | |
| 2 | 24 | | | |

| | | | | |
|---|---|---|---|---|
| | | 2 | | |
| | 5 | | 1 | |
| 17 | 24 | | | |

| | | | | |
|---|---|---|---|---|
| | | 5 | | |
| | 2 | | 1 | |
| 17 | 24 | | | |

| | | | | |
|---|---|---|---|---|
| | | 1 | | |
| | 2 | | 5 | |
| 17 | 24 | | | |

| | | | | |
|---|---|---|---|---|
| | | 1 | | |
| | 2 | | 5 | |
| 17 | 24 | | | |

**Ra array là**

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

## 8. Merge Sort

- **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Chia mảng thành 2 mảng con, tiếp tục lặp lại việc này ở các mảng con đã chia. Sau đó gộp các mảng nhỏ thành mảng lớn hơn đã được sắp xếp, gộp lần lượt tới khi tạo thành mảng ban đầu đã được xếp
- **Mã giả:**

```
Function Merge (a[], from, mid, to)
        first1 = form, last1 = mid
        first2 = mid + 1, last2 = to
        index = from
        aux = [to + 1]
        While ( first1 <= last1 and first2 <= last2) do
            If (a[first1] <= a[first2])
                aux[index] = a[first1]
                 index+=1
                 first1+=1
            Else
                aux[index] = a[first2]
                index+=1
                first2+=1
            Endif
         Endwhile
         While (first1 <= last1) do
              aux[index] = a[first1]
               index+=1
               first1+=1
          Endwhile
          While (first2 <= last2) do
              aux[index] = a[first2]
               index+=1
```

```
            first2+=1
        Endwhile
        For i = from to to do
            a[i] = aux[i]
        Endfor
```

```
Begin (a[], from, to)
If (from < to)
    mid = (from + to)/2
    MergeSort(a, from, mid)
    MergeSort(a, mid + 1, to)
    Merge(a, from, mid, to)
Endif
End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(nlog(n)) | O(nlog(n)) | O(nlog(n)) |
| Stable | Stable | | |

- **VD đơn giản với mảng 5 phần tử:**

**Mid=(left+right)/2**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1: Chia mảng thành các mảng con nhỏ hơn tại mid**

**Mid=2**

| 2 | 5 | 1 | | 24 | 17 |
|---|---|---|---|---|---|

**Mid=1** **Mid=3**

| 2 | 5 | | 1 | | 24 | | 17 |
|---|---|---|---|---|---|---|---|

**Mid=0**

| 2 | | 5 | | 1 | | 24 | | 17 |
|---|---|---|---|---|---|---|---|---|

**B2: Hợp các mảng con lại với nhau**

| 2 | 5 | | 1 | | 24 | | 17 |
|---|---|---|---|---|---|---|---|

| 1 | 2 | 5 | | 17 | 24 |
|---|---|---|---|---|---|

| 1 | 2 | 5 | 17 | 24 |
|---|---|---|---|---|

## 9. Quick Sort

- **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Chọn một phần tử trong mảng là pivot, từ đó chia các phần tử nhỏ hơn pivot sang 1 bên, bên còn lại là các phần tử lớn hơn pivot. Thực hiện tương tự với các mảng con của mảng ban đầu. Cứ chỉ nhỏ tiếp tới khi mảng được xếp
- **Mã giả:**

  Begin (a[], first, last)

```
pivot = a[(first + last)/2]
i = first, j = last
While (i <= j) do
        While (a[i] < pivot) do          // Duyệt đầu bên trái tới khi gặp phần tử đầu
            i = i + 1                    //nằm sai vị trí so với pivot
        Endwhile
        While (a[j] > pivot) do          // Duyệt đầu bên phải tới khi gặp phần tử đầu
            j = j – 1                    //nằm sai vị trí so với pivot
        Endwhile
        If (i <= j)
            swap(a[i], a[j])
            i = i + 1
            j = j – 1
        Endif
Endwhile
If (first < j)          // sau khi kết thúc vòng while ta được j nằm bên trái còn i nằm bên phải
        QuickSort(a, first, j)
Endif
If (i < last)
        QuickSort(a, i, last)
Endif
End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Phân đoạn hoàn hảo | Phân đoạn cân bằng | Phân đoạn không cân bằng, pivot nằm ở 1 trong đầu dãy |
| Độ phức tạp | O(nlog(n)) | O(nlog(n)) | $O(n^2)$ |
| Stable | không stable | | |

- **VD đơn giản với mảng 5 phần tử:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1: Chọn pivot =a[2].Sau đó chạy vòng while lần 1 được i=0, j=2. Swap a[i] ,a[j] tiếp đó i++, j—được i=j=1**

| | | | | |
|---|---|---|---|---|
| 1 | 5 | 2 | 24 | 17 |

**B2: Chạy vòng while lần 2 được j=0, i=1 => kết thúc while. Đệ quy lại quick sort cho 2 mảng a[0] đến a[j] với j=0 (không thay đổi gì) và a[i] với i=1 đến a[4] (pivot = a[2]).**

**Chạy vòng while được i=1, j=2 , swap và i++, j—được i=2, j=1 dừng while**

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 24 | 17 |

**B3: Đệ quy lại quick sort cho từ a[1] đến a[j] với j=1 ( không đổi ) và từ a[i] với i=2 đến a[4] ( pivot=a[3])**

**Chạy vòng while được i=3, j=4, swap và i++, j—được i=4, j=3 dừng while**

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

## 10. Counting Sort

- **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Đầu tiên đếm số lần xuất hiện của các phần tử trong mảng A và lưu vào mảng C, sau đó thay đổi chỉ số giới hạn mảng C. Cuối cùng duyệt từng phần tử của A vào mảng B chứa kết quả sắp xếp thông qua mảng C.

- **Mã giả:**

```
Begin (a[], n)
output[n]
max = a.max(), min = a.min()
range = max – min + 1
count[range]={0}

For i = 0 to n-1 do
        count[a[i] - min] = count[a[i] – min] + 1
Endfor

For i = 1 to range-1 do
        count[i] = count[i] +  count[i + 1]
Endfor

For i = 0 to n do
         output[count[a[i] – min] – 1] = a[i]
         count[a[i] - min] = count[a[i] – min] – 1
Endfor
```

```
For i = 0 to n do
        a[i] = output[i]
Endfor
End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự ngẫu nhiên | Mảng bị đảo ngược thứ tự |
| Độ phức tạp | O(n) | O(n) | O(n) |
| Stable | Stable | | |

- **VD đơn giản với mảng 5 phần tử:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1**: Đếm số lần xuất hiện của từng phần tử trong mảng a. Kết quả lưu vào mảng count

Max = 24

Min = 1

K = max – min + 1 = 24

Tại mảng count :

| Giá trị trong arr | 1 | 2 | ... | 5 | ... | 17 | .... | 24 |
|---|---|---|---|---|---|---|---|---|
| Giá trị trong count | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**B2**: Sửa đổi giá trị mảng count thể hiện vị trí đứng của nhóm các phần tử cùng giá trị a[i] khi sắp xếp

Mảng count

| Giá trị ứng với a[i] | 1 | 2 | ... | 5 | ... | 17 | .... | 24 |
|---|---|---|---|---|---|---|---|---|
| G.Trị trong count | 1 | 2 | 2 | 3 | 3 | 4 | 0 | 5 |

**B3**: Duyệt từng phần tử của mảng a và đặt nó vào đúng vị trí trong mảng output dựa vào mảng count

```
output[count[a[i] - min] - 1] = a[i];
count[a[i] - min]--;
```

Mảng count:

| Giá trị ứng với a[i] | 1 | 2 | ... | 5 | ... | 17 | .... | 24 |
|---|---|---|---|---|---|---|---|---|
| G.Trị trong count | 1 | 2 | 2 | 3 | 3 | 4 | 0 | 5 |

**3.1** Mảng output:

| 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|

Duyệt a[4]=17 => output[ count[17-1] -1]=a[4] , tức gán output[3]=17 và count[16]-=1

Mảng count thành

| Giá trị ứng với a[i] | 1 | 2 | ... | 5 | ... | 17 | .... | 24 |
|---|---|---|---|---|---|---|---|---|
| G.Trị trong count | 1 | 2 | 2 | 3 | 3 | 3 | 0 | 5 |

**3.2** Mảng output:

| 0 | 0 | 0 | 17 | 0 |
|---|---|---|---|---|

Duyệt a[3]=24 => output[ count[24-1] -1]=a[3] , tức gán output[4]=24 và count[23]-=1

Mảng count thành

| Giá trị ứng với a[i] | 1 | 2 | ... | 5 | ... | 17 | .... | 24 |
|---|---|---|---|---|---|---|---|---|
| G.Trị trong count | 1 | 2 | 2 | 3 | 3 | 3 | 0 | 4 |

**3.3** Mảng output:

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 17 | 24 |

Duyệt a[2]=1 => output[ count[1-1] -1]=a[2] , tức gán output[0]=1 và count[0]-=1

Mảng count thành

| Giá trị ứng với a[i] | 1 | 2 | ... | 5 | ... | 17 | .... | 24 |
|---|---|---|---|---|---|---|---|---|
| G.Trị trong count | 0 | 2 | 2 | 3 | 3 | 3 | 0 | 4 |

**3.4** Mảng output:

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 0 | 17 | 24 |

Duyệt a[1]=5 => output[ count5-1] -1]=a[1] , tức gán output[2]=5 và count[4]-=1

Mảng count thành

| Giá trị ứng với a[i] | 1 | 2 | ... | 5 | ... | 17 | .... | 24 |
|---|---|---|---|---|---|---|---|---|
| G.Trị trong count | 1 | 2 | 2 | 2 | 3 | 3 | 0 | 4 |

**3.5** Mảng output:

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 5 | 17 | 24 |

Duyệt a[0]=2 => output[ count[2-1] -1]=a[0] , tức gán output[1]=2 và count[1]-=1

Mảng count thành

| Giá trị ứng với a[i] | 1 | 2 | ... | 5 | ... | 17 | .... | 24 |
|---|---|---|---|---|---|---|---|---|
| G.Trị trong count | 1 | 1 | 2 | 3 | 3 | 3 | 0 | 4 |

**Ta được mảng output như sau và gán vô mảng a[] ban đầu**

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

## 11. Radix Sort

- **Ý tưởng chung:** : Giả sử sắp xếp mảng tăng dần. Radix Sort so sánh từng chữ số với nhau từ chữ số hàng bé nhất (hàng đơn vị) đến chữ số hàng lớn nhất.

- **Mã giả:**

```
Function findMax(a[], n)
    max = a[0]
    i
    For i = 1 to n do
        If (a[i] > max)
            max = a[i]
        Endif
    Endfor
    max

Function countSort( a[], n, digit)
    A[n]
    count[10]={0}
    i
    For i = 0 to n do
        count[(a[i] / digit) % 10 ] = count[(a[i] / digit) % 10 ] +1
    Endfor
    For i = 1 to max do
        count[i] = count[i] + count[i-1]
    Endfor
    For i = n – 1 to 0 do
        A[count[(a[i] / digit) % 10] – 1 ] = a[i]
        count[(a[i] / digit) % 10] = count[(a[i] / digit) % 10 ] – 1
    Endfor
    For i = 0 to n do
```

```
            a[i] = A[i]
        Endfor
```

```
Begin (a[], n)
digit
max = findMax(a, n)
For digit = 1 and max/digit > 0 and digit = digit *10
    countSort(a, n, digit)
Endfor
End
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Cơ số b nhỏ nhất và bằng 0 | K là phần tứ lớn nhất có thể có, b là cơ số | Giá trị b lớn và b = n |
| Độ phức tạp | $O(n\log_b n)$ | $O((n+b) * \log_b(k))$ | O(n) |
| Stable | Stable | | |

- **VD đơn giản với mảng 5 phần tử:**

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 2 | 5 | 1 | 24 | 17 |

**B1**: Tìm phần tử lớn nhất trong mảng : a[3] = 24 có 2 chữ số. VÌ vậy sẽ cần dùng 2 vòng lặp để sắp xếp.

**B2:** Lặp lần thứ 1

Dùng hàm countSort xép chữ số hàng đơn vị được

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 24 | 5 | 17 |

**B3** : Lặp lần thứ 2

Dùng countSort để sắp xếp các chữ số hàng chục

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

## 12. Flash Sort

- **Ý tưởng chung:** Giả sử sắp xếp mảng tăng dần. Gồm có 3 bước. Bước 1 là Phân lớp dữ liệu dựa trên giả thiết, bước 2 là Hoán vị toàn cục ( đưa các phần tử về lớp của chúng), bước 3 là Sắp xếp bố cục ( sắp xếp trong phạm vi từng lớp).

- **Mã giả:**

```
Begin (a[], n)
min = a[0], max = 0
m = 0.45*n
A[m]
For i = 0 to m do
    A[i] = 0
Endfor
For i = 1 to n do
    If (a[i] > min)
        min = a[i]
    Endif
    If (a[i] > a[max])
        max = i
    Endif
Endfor
If ( a[max] == min)
Endif
c = (m-1)/(a[max] – min)
For i = 0 to n do
```

```
        k = c*(a[i] – min)
        A[k] = A[k] + 1
    Endfor
    For i = 1 to m do
        A[i] = A[i] + A[i-1]
    Endfor
    swap(a[max], a[0])
    nmove = 0
    j = 0, t = m – 1, k = 0
    flash
    While (nmove < n – 1) do
        While ( j > A[k] – 1) do
            j = j + 1
            k = c*(a[j] – a[min])
        Endwhile
        flash = a[j]
        If ( k < 0 ) Endif
        While ( j != A[k]) do
            k = c*(flash – min)
            hold = a[t = (A[k] = A[k] – 1)]
            a[t] = flash
            flash = hold
            nmove = nmove + 1
        Endwhile
    Endwhile
    insertionSort(a,n)
    End.
```

- **Nhận xét:**

| | Trường hợp tốt nhất | Trường hợp trung bình | Trường hợp xấu nhất |
|---|---|---|---|
| Trường hợp | Mảng đã được sắp xếp | Mảng có thứ tự lộn xộn | Mảng có thứ tự đảo ngược |
| Độ phức tạp | O(n) | O(n) | O(n) |
| Stable | Không stable | | |

- **VD đơn giản với mảng 5 phần tử:**

a[0] a[1] a[2] a[3] a[4]

| 2 | 5 | 1 | 24 | 17 |
|---|---|---|---|---|

m = (int) 0.45 x 5 = 2

min = 1

max = 24

c = (m-1)/ (a[max] – min) =1/23

**B1**: Chia làm 2 lớp, lớp 0 gồm 3 phần tử [2, 5, 1], lớp 1 gồm [24, 17]

A = [3 2] ▪ A = [3 5]

Đổi a[max] và a[0]:

Mảng a:

| 24 | 5 | 1 | 2 | 17 |
|---|---|---|---|---|

flash = 24

Mảng A[1]-- : A[3 5] ▪ A[3 4]

**B2**: nmove = 1

Đổi chỗ a[0] và a[4]

Mảng a:

| 17 | 5 | 1 | 2 | 24 |
|---|---|---|---|---|

flash = 17

Mảng A[0]--: A[3 4] ▪ A[2 4]

**B3**: nmove = 2

Đổi chỗ a[0] và a[3]

Mảng a:

| | | | | |
|---|---|---|---|---|
| 2 | 5 | 1 | 17 | 24 |

flash = 2

Mảng A[0]-- : A[2 4] ▪ A[1 4]

**B4**: nmove = 3

Đổi chỗ a[0] và a[2]

Mảng a:

| | | | | |
|---|---|---|---|---|
| 1 | 5 | 2 | 17 | 24 |

flash = 1

Mảng A[1]-- : A[1 4] ▪ A[1 3]

**B5**: nmove =4

Đổi chỗ a[0] và a[1]

Mảng a:

| | | | | |
|---|---|---|---|---|
| 5 | 1 | 2 | 17 | 24 |

flash = 5

nmove = 4 = n – 1 ▪ Stop

**B6**: Phân lớp gồm lớp 0[ 5 1 2], lớp 1[ 17 24]

Dùng insertion sort để sắp xếp lại mảng này.

Mảng kết quả :

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 5 | 17 | 24 |

# III.Kết quả

## A.File CSV

-State: Random

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Size | Selection | Insertion | BinaryInse | BubbleSo | ShakerSor | ShellSort | HeapSort | MergeSor | QuickSort | Counting | RadixSort | FlashSort |
| 2 | 1000 | 2 | 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 3000 | 12 | 8 | 11 | 39 | 1 | 0 | 1 | 2 | 1 | 0 | 0 | 0 |
| 4 | 10000 | 129 | 102 | 67 | 342 | 0 | 1 | 3 | 4 | 2 | 1 | 1 | 0 |
| 5 | 30000 | 1853 | 1069 | 837 | 5486 | 1 | 12 | 14 | 17 | 7 | 1 | 7 | 2 |
| 6 | 100000 | 20632 | 11672 | 8038 | 51466 | 4 | 43 | 49 | 54 | 22 | 4 | 22 | 7 |

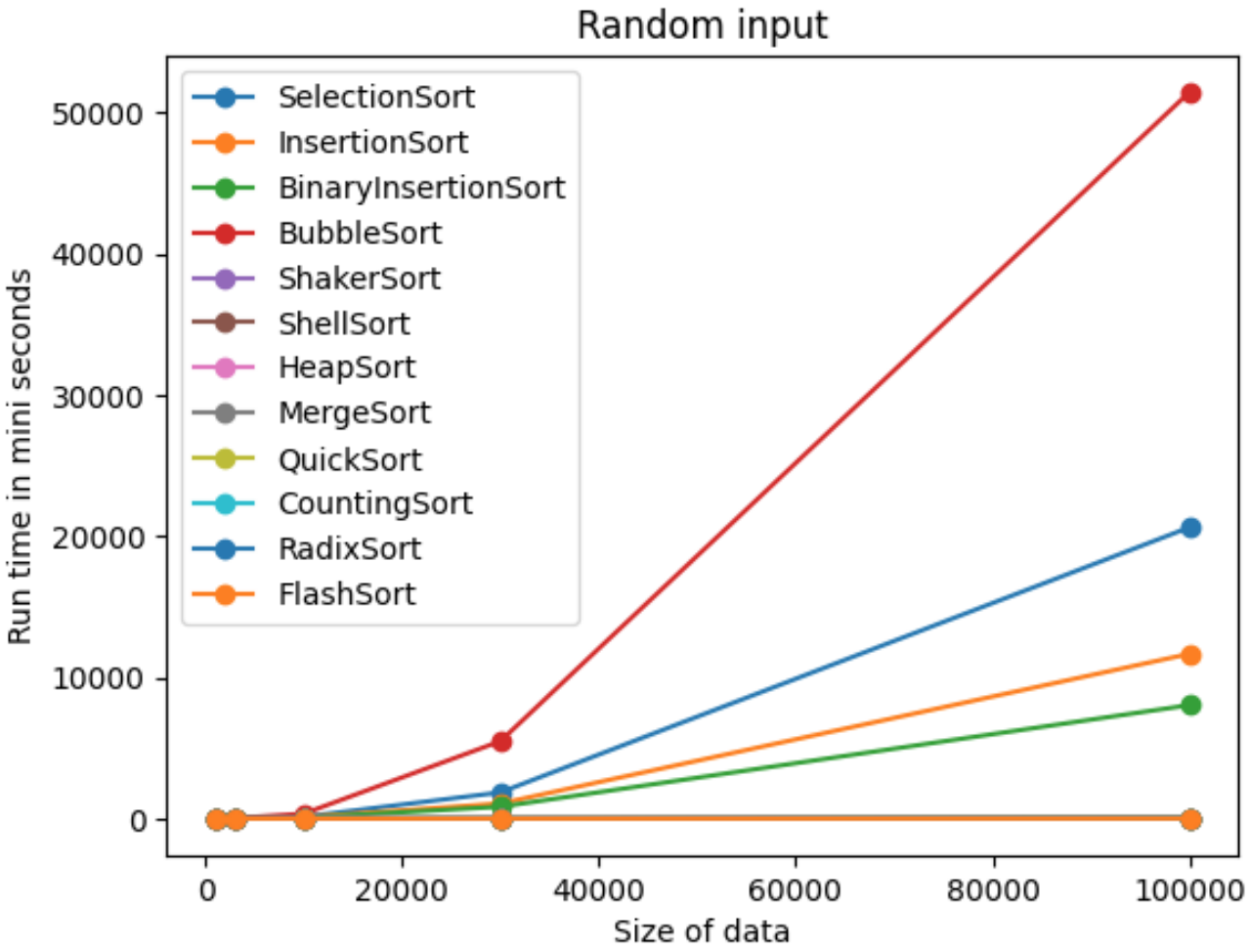
-State: Reversed

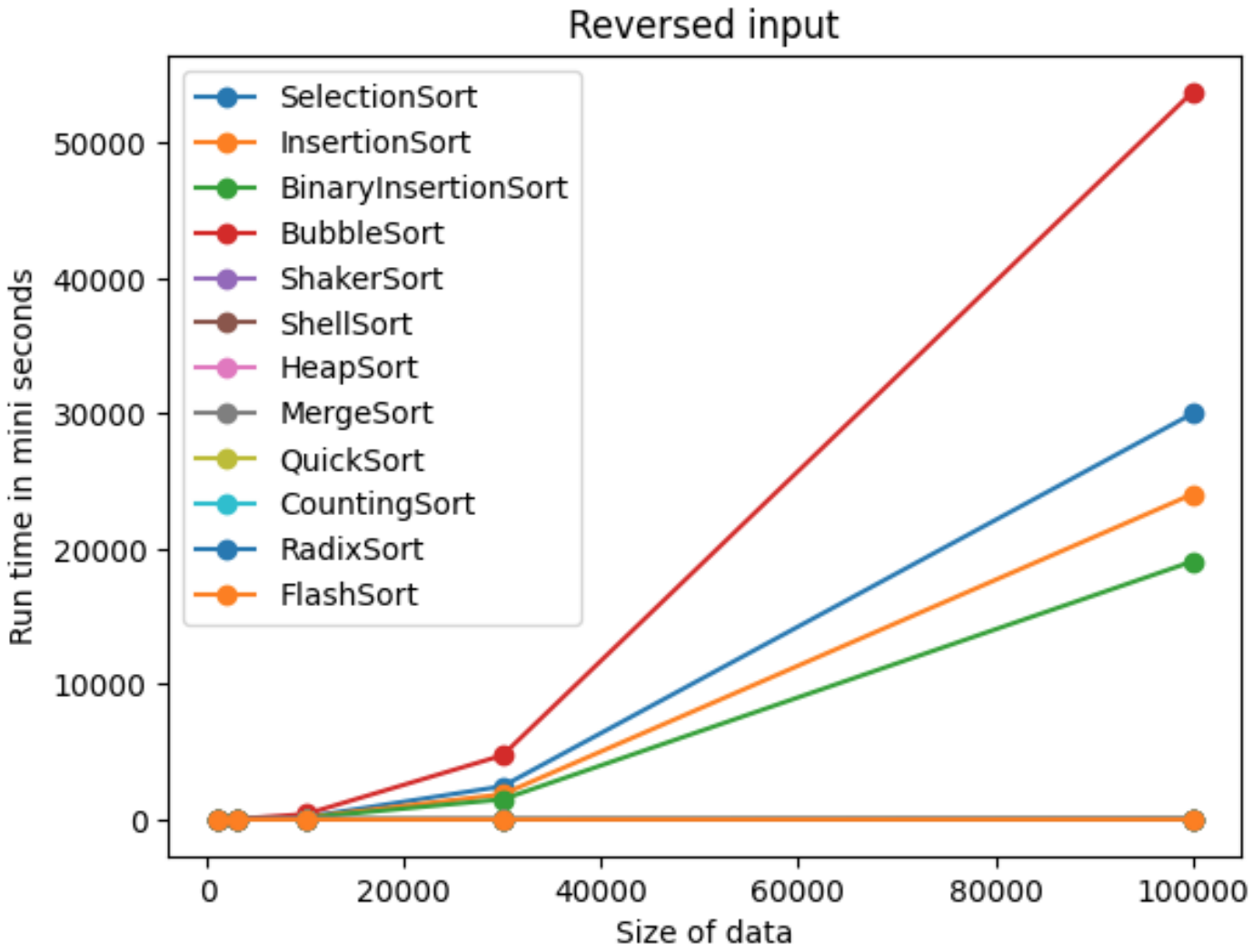
| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Size | Selection | Insertion | BinaryInse | BubbleSo | ShakerSor | ShellSort | HeapSort | MergeSor | QuickSort | Counting | RadixSort | FlashSort |
| 2 | 1000 | 1 | 2 | 2 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 3000 | 12 | 14 | 11 | 36 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 10000 | 174 | 145 | 122 | 389 | 0 | 1 | 2 | 3 | 0 | 0 | 1 | 0 |
| 5 | 30000 | 2451 | 1860 | 1485 | 4740 | 1 | 3 | 10 | 14 | 2 | 1 | 7 | 2 |
| 6 | 100000 | 29992 | 24034 | 19069 | 53748 | 2 | 14 | 34 | 40 | 7 | 3 | 17 | 5 |

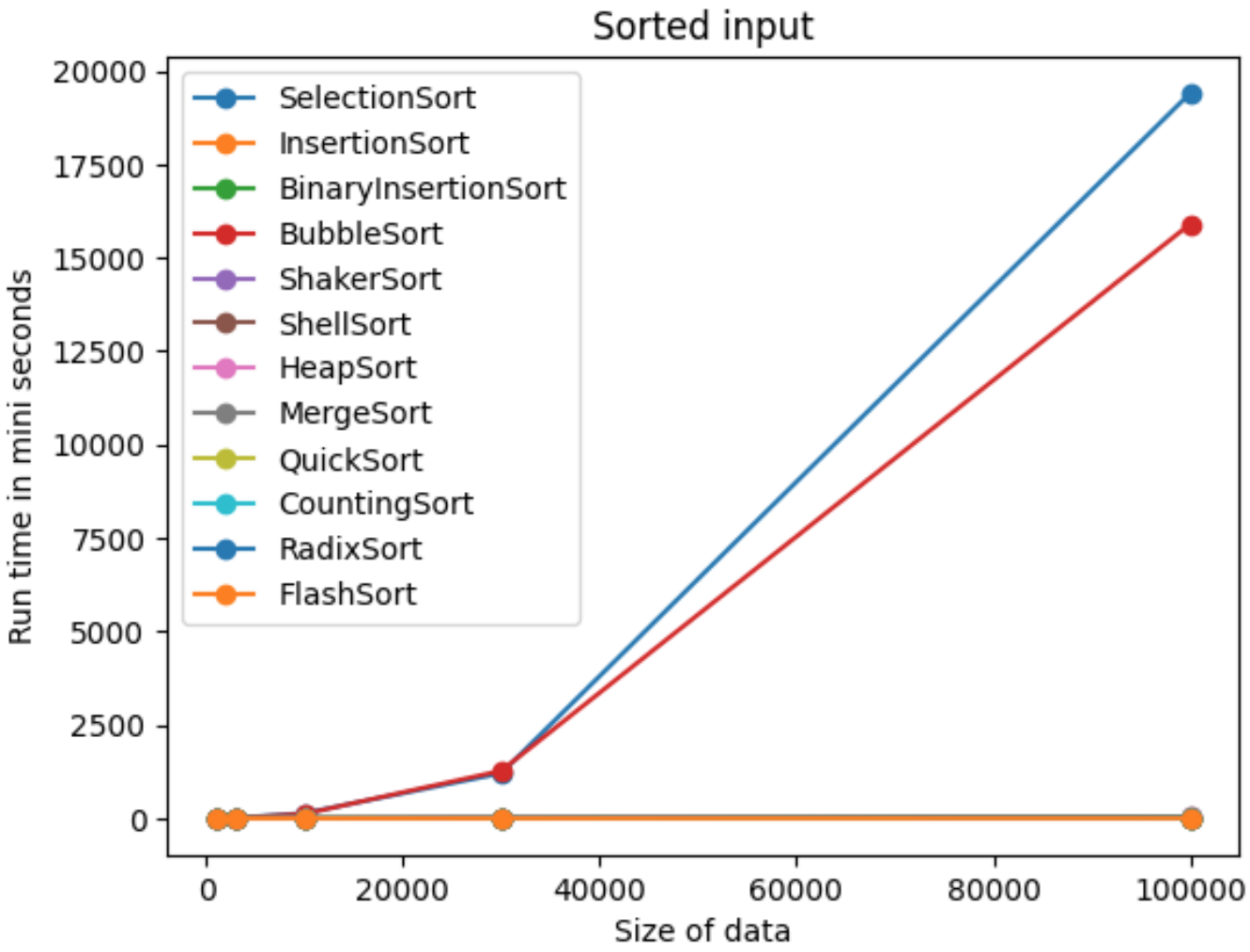
-State: Sorted

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Size | Selection | Insertion | BinaryInse | BubbleSo | ShakerSor | ShellSort | HeapSort | MergeSor | QuickSort | Counting | RadixSort | FlashSort |
| 2 | 1000 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 3000 | 11 | 0 | 0 | 10 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 10000 | 136 | 0 | 1 | 117 | 0 | 0 | 2 | 3 | 0 | 1 | 1 | 0 |
| 5 | 30000 | 1204 | 0 | 3 | 1270 | 0 | 2 | 10 | 15 | 1 | 1 | 5 | 2 |
| 6 | 100000 | 19401 | 0 | 13 | 15899 | 1 | 7 | 39 | 39 | 6 | 3 | 17 | 5 |

**B.Đồ thị**


Random input
SelectionSort
InsertionSort
BinaryInsertionSort
BubbleSort
ShakerSort
ShellSort
HeapSort
MergeSort
QuickSort
CountingSort
RadixSort
FlashSort
Run time in mini seconds
50000
40000
30000
20000
10000
0
0
20000
40000
60000
80000
100000
Size of data


Reversed input
SelectionSort
InsertionSort
BinaryInsertionSort
BubbleSort
ShakerSort
ShellSort
HeapSort
MergeSort
QuickSort
CountingSort
RadixSort
FlashSort
Run time in mini seconds
50000
40000
30000
20000
10000
0
0
20000
40000
60000
80000
100000
Size of data


Sorted input
Run time in mini seconds
Size of data
0
2500
5000
7500
10000
12500
15000
17500
20000
0
20000
40000
60000
80000
100000
SelectionSort
InsertionSort
BinaryInsertionSort
BubbleSort
ShakerSort
ShellSort
HeapSort
MergeSort
QuickSort
CountingSort
RadixSort
FlashSort

## C.Nhận xét

- Nhóm thuật toán O(n) và O(nlogn) chạy ổn định nhất với mọi trạng thái, mọi kích thước đã đưa ra thì thời gian chạy đều chỉ vài ms đến vài chục ms
- Kích thước dữ liệu càng lớn thời gian chạy của các thuật toán càng lâu, đặc biệt là nhóm thuật toán O(n^2) đại diện với Bubble sort là thuật toán chạy lâu nhất, khi mảng kích thước lên đến 100000 thời gian chạy của nó là khoảng 50000ms tại trường hợp trung bình và khoảng 55000ms với trường hợp tệ nhất. Kế là nhóm thuật toán O(nlogn) với thời gian chạy trung bình khoảng chỉ vài chục ms. Còn nhóm thuật toán O(n) thì chỉ vài ms
- Trạng thái giúp các thuật toán chạy nhanh nhất là sorted, khi hầu hết các thuật toán trường hợp tốt nhất đều là khi mảng đã được sắp xếp
- Trái lại trạng thái tệ nhất cho các thuật toán là reversed, khi khiên cho thời gian của các thuật toán chạy tăng lên đáng kể

## D. Tự đánh giá

- Tỷ lệ hoàn thành 100%

-Tự đánh giá 10đ

-1 số nguyễn tham khảo:

+GeeksforGeeks | A computer science portal for geeks

+https://howkteam.vn

+Introduction to Python (w3schools.com)

## IV.Bảng phân công công việc

| STT | Người phụ trách | Mô tả nội dung công việc | Bắt đầu | Kết thúc |
|---|---|---|---|---|
| 1 | Trần Đức Trung - 21280115 | .Tìm hiểu các thuật toán:<br>-Shaker sort<br>-Shell sort<br>-Heap sort<br>-Merge sort<br>-Quick sort<br>-Counting sort<br>. Cài đặt chương trình chạy các thuật toán với 3 kiểu dữ liệu và 5 kích thước dữ liệu. Đo thời gian chạy và vẽ trực quan các biểu đồ | 3/10/2022 | 24/10/2022 |
| 2 | Nguyễn Hoàng Thông-21280075 | .Tìm hiểu các thuật toán và cài đặt:<br>-Selection sort<br>-Insertion sort<br>-Binary insertion sort<br>-Bubble sort<br>-Radix sort<br>-Flash sort<br>-Binary search<br>-Linear search | | |


thank
you