Báo Cáo

1. **Các thuật toán sắp xếp**
   1. **Selection Sort**

-Ý tưởng : Chọn phần tử nhỏ nhất trong mảng rồi hoán đổi vị trí với phần tử thứ nhất và không quan tâm đến nó nữa, khi đó mảng còn lại n-1 phần tử rồi tiếp tục chọn phần tử nhỏ nhất trong n-1 phần tử đó rồi hoán đổi vị trí với phần tử thứ 2 cứ tiếp tục như thế cho đến hết mảng.

-Thuật toán:

+ Bước 1: Dùng vòng for lặp từ i=0 đến i=n-1

+ Bước 2: Tìm phần tử min (nhỏ nhất) từ a[i+1] đến a[n]

+ Bước 3: Hoán vị a[i] với a[min]

+ Bước 4: Lặp cho đến hết

-Đánh giá:

Số lần so sánh nhiều nhất là 3n(n-1)/2, Số lần so sánh trong trường hợp tốt nhất là n(n-1)/2

+Trường hợp tốt: O(n^2)

+Trường hợp xấu: O(n^2)

+Trường hợp trung bình: O(n^2)

* 1. **Merge sort**

-Ý tưởng: Chia mảng ban đầu thành 2 mảng con, tiếp tục lặp lại việc này ở các mảng đã chia cho đến khi nào mảng con còn 1 phần tử duy nhất. Sau đó so sánh 2 mảng con cùng cơ sở và ghép lại với nhau. Lặp lại việc này cho đến khi hết mảng cơ sở, ta sẽ được một mảng được sắp xếp hoàn chỉnh

-Thuật toán: Xây dựng hàm hàm MergeSort và Merge

Hàm MergeSort:

+Bước 1 : Chia mảng sắp xếp thành 2 nửa

+Bước 2 : Sắp xếp hai nửa đó bằng cách gọi đệ qui hàm MergeSort

+Bước 3 : Trộn 2 nửa đã được sắp xếp để được 1 mảng hoàn chỉnh

Hàm Merge:

+Bước 1: Tạo mảng L kích thước n1=m-l+1

+Bước 2: Tạo mảng R kích thước n2=r-m

+Bước 3: Mảng L[] lấy các phần tử từ a[0] đến a[m]

+Bước 4: Mảng R[] lấy các phần tử từ a[m] đến a[r-1]

+Bước 5: So sánh các phần tử ở mỗi mảng rồi đặt vào mảng cần sắp xếp

+Bước 6: Lặp lại bước 5 cho đến khi L và R duyệt xong

-Đánh giá:

+Trường hợp tốt nhất O(nlog(n))

+Trường hợp xấu nhất O(nlog(n))

+Trường hợp trung bình O(nlog(n))

* 1. **Heap Sort**

-Ý tưởng: Xây dựng mảng Max Heap rồi hoán đổi vị trị đầu tiên (Max) ra phía sau cùng, ta sẽ còn 1 mảng n-1 phần tử . Tiếp tục lặp lại thao tác trên cho đến khi mảng được sắp xếp hoàn chỉnh

-Mảng max heap thỏa mãn: Phần tử thứ i phải lớn hơn phần tử 2\*i+2 và 2\*i+2

-Thuật toán: Xây dựng 2 hàm Heapify và HeapSort

+Bước 1 : Dùng hàm Heapify để biến mảng thành một Heap

+Bước 2 : Dòng vòng for lặp từ i=n-1 đến i=0

+Bước 3: Đưa phần tử đầu tiên (MAX) về cuối dãy, tiếp tục dùng hàm Heapify tạo thành một Heap mới và quay lại bước 2

+Bước 4: Lặp cho đến khi sắp xếp hoàn chỉnh

-Đánh giá

+ Trường hợp xấu nhất: O(nlog(n)).

+Trường hợp tốt nhất: O(nlog(n)).

+ Trường hợp trung bình: O(nlog(n)).

* 1. **Quick Sort**

**-Ý tưởng:**QuickSort chia mảng thành hai danh sách bằng cách so sánh từng phần tử của danh sách với phần tử cuối cùng (được gọi là pivot). Những phần tử nhỏ hơn hoặc bằng phần pivot được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn chốt được đưa về phía sau và thuộc danh sách con thứ hai. Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1

-Thuật Toán: Xây dựng 2 hàm Quick Sort và Partition

Hàm Partition:

+Bước 1: chọn phần tử ở giữa mảng x=a[(l+r)/2], i=l, j=r.

+Bước 2: Phát hiện và hiệu chỉnh cặp phần tử arr[i], arr[j] sai vị trí:

+Bước 2.1: Trong khi (arr[i]x) giảm j.

+Bước 2.3: Nếu i<=j thì: Hoán vị arr[i], arr[j]. Tăng i và giảm j.

+Bước 3: So sánh i và j:

Nếu i<j thì lặp lại bước 2

Ngược lại thì dừng

Hàm QuickSort

Bước 1: Phân hoạch dãy ban đầu thành 3 dãy:

Dãy 1: arr1, arr2…arr(j) < x.

Dãy 2: arr(j+1)…arr(i-1) = x.

Dãy 3: arr(i)…arr(r) > x.

Bước 2: Sắp xếp:

Nếu i<j : Phân hoạch dãy a[i] và a[j]

Nếu i<r : Phân hoặc dãy a[j] và a[r]

-Đánh giá:

+Trường hợp xấu nhất : O(n^2)

+Trường hợp tốt nhất : O(nlog(n))

+Trường hợp trung bình : O(nlog(n))

* 1. **Bubble Sort**

-Ý tưởng: So sánh và đổi chỗ 2 số liền tiếp nhau (nếu đứng sai thứ tự), sau lần duyệt thứ nhất ta sẽ đưa được số lớn nhất ra cuối cùng. Lặp lại thao tác trên ta sẽ được dãy số hoàn chỉnh

-Thuật toán

+Bước 1: Dùng vòng for lặp từ i=0 đến i=n-1

+Bước 3: Dùng vòng for lặp từ j=0 đến j=n-i-1

+Bước 4: Hoán vị a[j] và a[j+1] nếu a[j]>a[j+1]

+Bước 5: Lặp lại cho đến khi duyệt hết mảng

-Đánh giá:

+Trường hợp xấu nhất : O(n^2)

+Trường hợp tốt nhất : O(n)

+Trường hợp trung bình O(n^2)

* 1. **Insertion Sort**

-Ý tưởng : Chèn mỗi phần tử vào dãy con đã được sắp xếp

-Thuật toán:

+Bước 1: Dùng vòng for lặp từ i=1 đến i=n-1

+Bước 2: Gán key=a[i]

+Bước 3: Tìm vị trí thích hợp trong mảng đã sắp xếp để chèn phần tử key vào

+Bước 4: Lặp lại cho đến khi hết mảng

-Đánh giá

+Trường hợp xấu nhất: O(n^2)

+Trường hợp tốt nhất : O(n)

+Trường hợp trung bình : O(n^2)

* 1. **Binary-Insertion Sort**

-Ý tưởng: Chèn mỗi phần tử vào dãy con đã được sắp xếp. Mà thao tác tìm kiếm vị trị cần chèn bằng Binary Search

-Thuật toán:

+Bước 1: Dùng vòng for lặp từ i=1 đến i=n-1

+Bước 2: Gán key=a[i]

+Bước 3: Tìm vị trí thích hợp trong mảng bằng Binary Search

+Bước 4: Chèn phần từ key vào vị trí thích hợp

+Bước 5: Lặp lại cho đến khi hết mảng

-Đánh giá:

+Trường hợp xấu nhất : Hàm BinarySearch sẽ phải tìm hết mảng nên độ phức tạp là O(n^2)

+Trường hợp tốt nhất : O(nlog(n))

1. **Các trường hợp khảo sát**
   1. **Generate Random Data**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Selection Sort | Merge Sort | Heap Sort | Quick Sort | Bubble Sort | Insertion Sort | Binary Insertion Sort |
| 3000 | 0.012 | 0.003 | 0.002 | 0.001 | 0.078 | 0.007 | 0.006 |
| 10000 | 0.125 | 0.007 | 0.009 | 0.002 | 0.911 | 0.065 | 0.053 |
| 30000 | 1.164 | 0.027 | 0.03 | 0.008 | 8.25 | 0.609 | 0.453 |
| 100000 | 12.432 | 0.127 | 0.138 | 0.032 | 103.85 | 8.269 | 4.369 |
| 300000 | 142.35 | 0.289 | 0.364 | 0.228 | 1154.2 | 72.325 | 50.145 |

Đánh giá: Thuật toán Seclection Sort và Bubble Sort chạy rất chạy

* 1. **Generate Sorted Data**
  2. **Generate Reverse Data**
  3. **Generate Nearly Sorted Data**

1. **Đánh giá và so sánh thuật toán**