Tên SV:Huỳnh Công Thức

MSSV:1613494

Chương 1:

Câu 3:

a) Trường hợp xấu nhất :

Phần tử lớn nhất nằm ở đầu:

Nên số phép só sánh sẽ là : n-1+n-1

C(n)=2n-2

b)Trường hợp tốt nhất:

Phần tử bé nhất nằm ở đầu sau đó đến trung và cuối cùng là lớn nhất:

Số phép so sánh sẽ là :n-1

C(n)=n-1

c)Trường hợp n=3

Mảng có 3 số :abc(lớn,trung,nhỏ)

Tổng có 6 hoán vị :

abc:số phép so sánh = 4

acb:số phép so sánh =4

bac:3

bca:3

cab:3

cba:2

Tổng phép so sánh là:19

Xác suất rơi vào 1 trường hợp là 1/6

Độ phức tạp: C(N)=19/6

Câu 4:

Nhập dữ liệu với kích thước n.

Với j=1 thì thao tác so sánh 1

Với j=1*b thì thao tác so sánh 2

Với j=b*b thì thao tác so sánh 3

…

Với $j=b^k$ thì thao tác so sánh là k+1

Trước khi kết thúc còn có thêm 1 thao tác so sánh nữa vậy nên cần cộng thêm 1.

C(N)=1+2+3+…+k+1+1

C(N)=n(n+1)/2+1

Câu 7:

$C_n=2C_{n/2}+N+1$ N>=2 $C_1=0$

$C_n=2+4+8+…+N/2+N+\log_2(N)=C_1(1-2^n)/(1-2)+\log_2(N)$

Câu 8:

C(n)=c+C(n-1) C(1)=d

C(n-1)=c+c+C(n-2)

=>C(n)=(N-1)c+d

Câu 11 :

$N=2^n$

$C_N=4C_{N/2}+N^2$

$C_N=2^6C(2^{n-3})+2^{2n}+2^{2n}+2^{2n}$

$C_N=2^{2(n)}.C(1)+n.2^{2n}=2^{2n}(1+n)\sim N^2(1+\log_2 N)$

Cau12 :

$N=2^n$

$C_N=2C_{N/2}+N^2$

$C_{N/2}=C_{N/4}+(N/2)^2$

$C_N=2^3C(2^{n-3})+2^{2n-2}+2^{2n-1}+2^{2n}$

$C_N=2^nC(2^0)+2^{n+1}+..+2^{2n-1}+2^{2n}$

$C_N=2^nC(1)+1+4+8+…+2^n+2^{n+1}+..+2^{2n-1}+2^{2n}-(1+4+8+…+2^n)$

$C_N=2^nC(1)+2^{2n+1}-1+2^{n+1}-1\sim 2^{2n}\sim O(N^2)$

Chương 2 :

```
Quicksort(left,right,arr):
    i=left
    j=right-1
    pivot=arr[right]
    if (left<right){
        while(true){
            while(i<j && arr[i]<pivot)i++;
            while(i<j && arr[j]>pivot)j--;
            if (i>=j) break;
            swap(&arr[i],&arr[j])
            i++;
            j--;
        }
        swap(&arr[i],&arr[right])
        Quicksort(arr,left,i+1)
        Quicksort(arr,i+1,right)
    }
```

Câu 13 :

a) Tìm ra khoảng cách nhỏ nhất giữa 2 phần tử liên tiếp

b) O(n)

Chương 3:

```
bruteforce(arr){
    min =arr[0]
    for (i=1;i<n;i++)
        if (arr[i]<min){
            min=arr[i]
        }
    return min
```

}

divide-conquer

```
merge(int arr[],int l,int m,int r){
    int n1 =m+1-l
    int n2 =r-m
for (int i=0;i<n1;i++){
    L[i]=arr[l+i]
}
for (int j=0;i<n2;i++){
    R[j]=arr[m+1-1+j]
}
int i=0,j=0,k=0;
while(i<n1 && j<n2)
{
    if (L[i]<R[j]){
        arr[k]=L[i]
        i++;
    }
    if (L[i]>R[j]){
        arr[k]=R[j]
        j++;
    }
    k++
}
while(i<n1){
    arr[k]=L[i]
    i++;
    k++;
}
while(j<n2){
    arr[k]=R[j]
    k++;
    j++;
}
}
mergesort(int arr[],int l,int r){
    int n=(l+r)/2
    mergesort(arr,l,n-1)
    mergesort(arr,n-1,r)
    merge(arr,l,n,r)
}
int divide-conquer(arr){
merge(arr,0,n)
return arr[0]
```

Câu 5:

Hiệu chỉnh DFS để phát hiện chu trình:

Để phát hiện được chu trình trong đồ thị. tiến hành lưu vết các điểm đã duyệt (đã thêm vào stack) trong tiến trình sau có quay lại đỉnh đó thì tức là chu trình trong đồ thị.

```
class Graph():
    def __init__(self,vertices):
        self.graph = defaultdict(list)
        self.V = vertices

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def isCyclicUtil(self, v, visited, recStack):
        visited[v] = True
        recStack[v] = True

        for neighbour in self.graph[v]:
            if visited[neighbour] == False:
                if self.isCyclicUtil(neighbour, visited, recStack) == True:
                    return True
            elif recStack[neighbour] == True:
                return True

        recStack[v] = False
        return False

    def isCyclic(self):
        visited = [False] * self.V
        recStack = [False] * self.V
        for node in range(self.V):
            if visited[node] == False:
                if self.isCyclicUtil(node,visited,recStack) == True:
                    return True
        return False
```

Độ phức tạp O(V+E)