Taller 2

Análisis y diseño de algoritmos

Integrantes

Sebastian Peñaranda

2041138-3743

Luis Felipe Belalcazar

2020783-3743

Profesor

Jesús Aranda

Universidad del Valle

Escuela de ingeniería de sistemas y computación

Programa de ingeniería de sistemas (3743)

Cali, Colombia

Diciembre ,2023

1. Manejo de cola de atención

Idea de la solución:

Se creo un array interno con los diferentes puestos para guardar su tiempo de atención con este array de tamaño P[1<i<N], quedando la entrada para el ejercicio como:

N-> Número de clientes : 1> N <= 10^4

A->Tiempo de atención por persona Ai $(0 \leq i < N)$

T-> Tiempo máximo de atención $1 \leq T \leq 10^9$

La idea base fue en restar del array de puestos el tiempo de atención de cada cliente tal que P[i] - A[i], teniendo en cuenta que sea mayor a 0, para guardar el tiempo de atención que quedaba en cada puesto después de atender a un cliente, Ejemplo:

P[8,8,8]

A[4,2,8] ➡️ A[i] =4 P[i]=8 ➡️ P[i] - A[i] = 4 ➡️P[4,8,8]

En este caso ya habíamos atendido al primer cliente y cambiaríamos al que sigue, cambiando de posición en A y solo cambiando de puesto si el primer o siguientes puestos no tienen tiempo suficiente para atender al nuevo cliente:

P[4,8,8]

A[4,2,8] ➡️ A[i+1] =2 P[i]=4 ➡️ P[i] - A[i] = 2 ➡️P[2,8,8]

Por lo que se cambia a un nuevo cliente:

P[4,8,8]

A[4,2,8] ➡️ A[i+2] =8 P[i+1]=8 ➡️ P[i] - A[i] = 0 ➡️P[2,0,8]

Ya cuando se atiende a todos los clientes, solo queda contar cuántos puestos con diferente valor del T (Tiempo máximo de atención) hay en el array P, en este caso P[2,0,8] serían 2 por lo que el número mínimo de puestos para atención de clientes de forma simultánea es 2.

Análisis de Complejidad:

Para este ejercicio se entiende que el peor caso sería el caso promedio con un arreglo de nombre tiempoA[N] , P[N] donde N= 10^4, y cada cliente del arreglo tenga un tiempo promedio de 1 hasta T, (con tiempo máximo T de atención x), recorriendo el arreglo de puestos hasta n^2 veces que sea necesario, ya que necesita verificar si en algún puesto anterior se encuentra con el tiempo de atención restante para atenderlo

Pseudocodigo:

```
int N
int T = 8
arreglo P[N] = [T,T,T…]
arreglo A[N]
while N > 0
   while P[i] - A[j] >= 0
          P[i]  = P[i] - A[j]
           N- -
           j++
           i = 0
    i++
int contadorFinal
for x in range (0, personas,1)
    if (P[x] != T)
         contador Final++
return contadorFinal
```

Como enfoque tomamos en cuenta el insertion sort, modificándolo ya que en vez de hacer verificaciones de orden hacemos una verificación de sumas donde si el puesto actual no tiene suficiente tiempo para atender al cliente, cambiamos de posición en el array de puestos significando que se necesita un puesto adicional cada vez que el puesto anterior no logre atender a un cliente para lograr la simultaneidad, y si la condicional del segundo while se cumple asigna el valor de la resta a P[i] y reinicia las posiciones de P ya que cada siguiente cliente tiene que recorrer cada puesto anterior para verificar si hay un puesto que pueda atenderlo, ya que se busca la cantidad mínima de puestos y se atiende en orden de llegada, al tener que recorrer el array una y otra vez se comporta como el peor caso del insertion sort solo que en este caso sería en realidad el caso promedio algoritmo tiene un comportamiento $O(n^2)$.

2. Selección de apartamentos

Idea de la solución:

La entrada para este ejercicio sería :
m: -> Número de apartamentos
n: -> Número de aplicantes
k: -> Tolerancia en el precio de los aplicantes
M: -> Precios de los apartamentos
N: -> Dinero tentativo de los aplicantes

La idea tras la solución de este ejercicio es evaluar cada dinero de los postulantes (N) con sus respectivas tolerancias en cada Precio de los apartamentos (M), cuando encuentra un comprador este agrega un contador a la cantidad de apartamentos alquilados y omitiendo los apartamentos y comprados que ya se hayan involucrado en la compra, para ya finalizar entregando la cantidad de apartamentos alquilados en totalidad.

Una manera para que la complejidad no sea tan alta a la hora de tener muchos apartamentos(m) y aplicantes (n) es organizar los precios de los apartamentos (M) y el dinero (N) de manera ascendente, dando el caso en el que cuando se esté recorriendo la lista (N) y el valor del aplicante más su tolerancia se exceda al precio actual del apartamento que se está comparando, este pase directamente al siguiente apartamento ya que los precios siguientes no podrán hacer la compra del apartamento.

Este proceso de ordenación se podrá realizar de muchas maneras, pero la más convencional sería el merge sort, ya que no los datos no se encuentran de alguna manera cuasi ordenados, ni tienden a algún patrón ya ordenado.

Análisis de Complejidad:

Se utilizó el merge sort para ordenar el precio de los apartamentos y el monto de dinero tentativo de los aplicantes para poder encontrar un rango de precios para cada cliente y no tener que recorrer todo el array buscando por lo cual esta parte del código tendría complejidad O(n log n) como mínimo.

El resto del algoritmo se comporta en el peor de los casos como $O(n^2)$, ya que el peor de los casos seria donde los aplicantes no puedan comprar ningún apartamento y el primer valor de los apartamentos supere a todos los aplicantes, ejemplo:

K=5
M[100, 120 ,130]
N[5,10,20,30,40,50,60,70,80,90]

Ya que 100 supera a todos los valores de los aplicantes entonces cada valor de los aplicantes tendrían que recorrer el array de los precios todas las veces siendo muy ineficiente, y por tanto no logrando hacer el filtrado de los datos, y evitando llegar a un orden $O(n^2)$ como en el siguiente ejemplo.

M[50, 70 ,130]
N[5,10,20,30,40,60,70,80,90]

hace la validacion en el 40 y los datos resaltados en amarillo se descartan ya que se rompe el ciclo y se borra el apartamento alquilado y su comprador para volver a buscar el nuevo valor del siguiente apartamento, para no gastar recursos ni tiempo filtrando los datos siendo asi el codigo mucho mas eficiente, siendo asi el caso promedio final O(nlog n) y el peor caso del algoritmo $O(n^2)$.

Pseudocodigo:

```
int m
int n
arreglo M[m]
arreglo N[n]
for i in range m
        for j in range n
            if ((N[j] - k <= M[i] <= N[i] + k) ||  M[i]== N[j]))
                contador++
            if  N[j + i] - k > M[i]
                break;
return contador;
```

3. Mesas ocupadas

Idea de la solución:

Las entradas para esta solución:
- n: -> Número de grupos que entraron y salieron del restaurante.
- T: -> Arreglo conformado por partes que representa el momento en el que el i-ésimo grupo entró y salió del restaurante.

La idea tras la solución de este ejercicio es encontrar la mayor cantidad de mesas utilizadas en el mismo instante de tiempo, así pues se idea una manera de conseguir el intervalo de tiempo (I) en la que las mesas están siendo ocupadas, encontrando primero el menor tiempo de entrada (E) de los grupos que entraron y el mayor tiempo de salida de los grupos (S) y así haciendo una diferencia entre E y S.

Ya con un intervalo de tiempo (I) apartir de aqui nos inspiramos en el ordenamiento counting sort donde necesitamos contar cada una de las mesas en los distintos instante de tiempo del intervalo I, donde se guarda en un array llamada cantidad de mesas(M) los resultados y retornando la mayor cantidad de mesas del intervalo de tiempo.

Análisis de Complejidad:

Para empezar se tomó en cuenta el algoritmo de ordenamiento counting sort por tanto mínimamente la complejidad es de O(n) ya que este ordenamiento es lineal, siguiendo el pseudocódigo se tiene:

Pseudocodigo:

```
int n
int I
arreglo T[n]
arreglo E[n]
arreglo S[n]
arreglo M[I]
int totalMesas
for i in range I
        for j in range n
            if (E[j] <= i <= S[j])
                M[i] += 1
            if  i == 0
                totalMesas = M[i]
            if  i > 0
                if totalMesas < M[i]
                        totalMesas = M[i]
return totalMesas
```

En este algoritmo podemos decir que nuestro peor caso es que no haya valores en los cuales sean demasiado altos ya que aumenta la complejidad a $O(n^2)$ ya que el primer bucle se realiza según la cantidad de intervalos en las que las mesas se encuentran ocupadas dando un total de O(n) y el segundo bucle se repite la cantidad de grupos que haya, dando $n*n = n^2$, siendo asi la complejidad total $O(n^2)$.
En el caso promedio, donde los valores sean menores, este algoritmo solo alcanza una complejidad O(n), ya que en el segundo bucle se comporta más como una operación lineal, como ejemplo : 3n, 6n, 11n,