Programación y Algoritmos I

Tarea 6

Marco Antonio Esquivel Basaldua

Problema 1

En el peor de los casos los elementos en A están ordenados de mayor a menor. En este caso la complejidad del algoritmo es $O(n^2)$ ya que el ciclo for se debe ejecutar N veces .

En el mejor de los casos los elementos en A ya están ordenados. En este caso la complejidad es $O(n)$ ya que el ciclo for se ejecuta una sola vez yendo de 0 a $N-1$.

Problema 2

1. Para encontrar $T(n)$:
    - Se realizan dos asignaciones
    - Cada vez i se divide entre 2 en división entera (equivalente a $i >>= 1$). Ya que i toma inicialmente el valor de n y el ciclo $while$ es válido mientras i sea mayor que 0, se tiene $log_2(n)$ iteraciones, en cada una de ellas hay dos asignaciones.

      $T(n)$ se calcula como: $T(n) = 2 + 2 * log_2(n)$

      La complejidad expresada en $O()$ es $O(log(n))$
2. Para encontrar $T(n)$:
    - Se realizan dos asignaciones
    - los valores de s van cambiando dentro del ciclo $while$ para después de la iteración j de la forma:
        - $s_0 = 1$
        - $s_1 = 3$
        - $s_2 = 6$
        - $s_3 = 10$
        - $s_j = s_{j-1} + (j+1)$

      Los valores que toma s son los números triangulares, entonces: $s_j = \frac{(j+1)(j+2)}{2}$

      Comprobación por inducción:

$$s_j = \frac{(j+1)(j+2)}{2}$$

$$s_{j+1} = s_j + (j+2)$$

$$= \frac{(j+1)(j+2)}{2} + (j+2)$$

$$= \frac{(j+1)(j+2) + 2(j+2)}{2}$$

$$= \frac{(j+2)(j+1+2)}{2}$$

$$s_{j+1} = \frac{(j+2)(j+3)}{2}$$

Con esto se comprueba la hipótesis.

- Supongamos que se realizan j iteraciones. Expresamos j en función de n

$$s_j \leq n$$
$$\frac{(j+1)(j+2)}{2} \leq n$$
$$j^2 + 3j + 2 \leq 2n$$
$$j^2 + 3j + 2 - 2n \leq 0$$
$$j^2 + 3j + 2(1-n) \leq 0$$
$$j \leq \frac{-3 \pm \sqrt{9 - 8(1-n)}}{2}$$
$$j \leq \frac{-3 \pm \sqrt{1+8n}}{2}$$

Se toma solo el valor positivo. $T(n)$ se expresa como:

$$T(n) \leq 2 + \frac{-3 + \sqrt{1+8n}}{2}$$

Ya que el término que contiene a n es $\frac{\sqrt{1+8n}}{2}$, la complejidad en notacion $O()$ es: $O(\sqrt{n})$

Problema 3

Para la realización de este árbol se evitará trabajar con estructuras, para eso se guardan los datos ingresados al árbol en un arreglo dinámico comenzando desde el indice 1. En el indice 0 se registrará la cantidad de datos presentes en el árbol, este valor aumenta en uno si se ingresa un dato y disminuye en uno si se remueve el valor máximo.

Pseudo-código:

- Pide al usuario definir la cantidad máxima de nodos en el árbol.
- Desplegar el menú: 1 para ingresar dato, 2 para remover máximo, 3 para terminar
  - Si se presiona 1:
    - Escribir dato a ingresar al árbol
  - Si se presiona 2:
    - Mostrar el valor máximo en el árbol
  - Si se presiona 3:
    - Salir del menú
- End

Ejemplo

Al ingresar los valores uno por uno $2, 4, 15, -2, 0, 1, 23, 20$ en un árbol de 9 nodos y después removerlos por prioridad se obtienen los siguientes valores paso a paso (recordar que el primer valor indica cuántos datos existen dentro del árbol):

- Se ingresa 2
  - Resultado: 1 2 0 0 0 0 0 0 0 0
- Se ingresa 4
  - Resultado: 2 4 2 0 0 0 0 0 0 0
- Se ingresa 15
  - Resultado: 3 15 2 4 0 0 0 0 0 0

- Se ingresa -2
    - Resultado: 4 15 2 4 -2 0 0 0 0 0
- Se ingresa 0
    - Resultado: 5 15 2 4 -2 0 0 0 0 0
- Se ingresa 1
    - Resultado: 6 15 2 4 -2 0 1 0 0 0
- Se ingresa 23
    - Resultado: 7 23 15 4 -2 0 1 2 0 0
- Se ingresa 20
    - Resultado: 8 23 15 20 -2 0 1 2 4 0
- Retirar dato mayor
    - Resultado: 7 20 15 4 -2 0 1 2 0 0
- Retirar dato mayor
    - Resultado: 6 15 2 4 -2 0 1 0 0 0
- Retirar dato mayor
    - Resultado: 5 4 2 1 -2 0 0 0 0 0
- Retirar dato mayor
    - Resultado: 4 2 0 1 -2 0 0 0 0 0
- Retirar dato mayor
    - Resultado: 3 1 0 -2 0 0 0 0 0 0
- Retirar dato mayor
    - Resultado: 2 0 -2 0 0 0 0 0 0 0
- Retirar dato mayor
    - Resultado: 1 -2 0 0 0 0 0 0 0 0

Observaciones y conclusiones

- Con esta implementación se evita el uso de estructuras y se aprovecha la posición $[0]$ del arreglo de datos que originalmente no se utiliza.
- Los datos en el arreglo son inicializados con 0, esto hace difícil de ver cuando algunos de los datos ingresados han sido 0, sin embargo se puede saber cuales son datos "válidos" revisando el primer valor del arreglo el cual indica cuantos valores existen dentro del árbol. Otra alternativa podría ser representar los espacios vacíos del árbol con NAN o hacer un arreglo de tamaño variable de acuerdo a la cantidad de datos ingresados y extraídos.
- El ingresar los datos uno por uno al árbol puede resultar algo tedioso, sin embargo es la mejor manera de ir analizando y validando el comportamiento del mismo.

Problema 4

Para la implementación de una $cola$ se hará uso de dos $pilas$, prácticamente se implementará una $cola$ sin recurrir a la programación de una. Para ello se utilizarán dos $pilas$, $pila1$ y $pila2$, de la siguiente manera (hay que recordar que el funcionamiento de una pila es bajo el funcionamiento $LIFO$ mientras que el de una cola es bajo el funcionamiento $FIFO$):

- Para ingresar un dato a la $cola$ simulada:
    - ingresar el dato a $pila1$, enseguida vaciar $pila1$ en $pila2$
- Para remover el primer dato en la $cola$ simulada:

- remover un dato de $pila2$, enseguida vaciar $pila2$ en $pila1$

Al igual que en la implementación del problema 3, se evitará hacer uso de estructuras. Se trabajará con dos arreglos dinámicos de tamaño $N+1$ que servirán como $pila1$ y $pila2$ y, al igual que en el problema 3, el dato en la posición [0] de cada uno de los arreglos indicará la cantidad de datos presentes en cada arreglo y por tanto en la $cola$ simulada. El dato N será solicitado al usuario al inicio de la ejecución.

Para visualizar el comportamiento de las $pilas$ en el programa, cada vez que se ingresa un nuevo dato o se solicita extraer uno de ellos se muestra por consola los datos presentes en $pila1$ y $pila2$.

Pseudo-código:

- Pide al usuario definir la cantidad máxima de datos en la cola.
- Desplegar el menú: 1 para ingresar dato, 2 para recuperar el primer dato en la cola, 3 para terminar
  - Si se presiona 1:
    - Escribir dato a ingresar a la $cola$
      - El dato se ingresa en $pila1$
      - $pila1$ se vacía en $pila2$
  - Si se presiona 2:
    - Mostrar el primer valor de la $cola$
      - Remover el último valor en $pila2$
      - $pila2$ se vacía en $pila1$
  - Si se presiona 3:
    - Salir del menú
- End

Ejemplo

Se van a ingresar los datos $2, 5, 7, -2, 4$ uno por uno a una $cola$ simulada de cinco posiciones, después serán removidos uno por uno.

- Se ingresa el valor 2
  - $pila1$= 1 2 0 0 0 0
  - $pila2$= 1 2 0 0 0 0
- Se ingresa el valor 5
  - $pila1$= 2 2 5 0 0 0
  - $pila2$= 2 5 2 0 0 0
- Se ingresa el valor 7
  - $pila1$= 3 2 5 7 0 0
  - $pila2$= 3 7 5 2 0 0
- Se ingresa el valor -2
  - $pila1$= 4 2 5 7 - 2 0
  - $pila2$= 4 -2 7 5 2 0
- Se ingresa el valor 4
  - $pila1$= 5 2 5 7 -2 4
  - $pila2$= 5 4 -2 7 5 2
- Se extrae un valor

- Se obtiene 2
- $pila1$= 4 5 7 -2 4 0
- $pila2$= 4 4 -2 7 5 0

- Se extrae un valor
  - Se obtiene 5
  - $pila1$= 3 7 -2 4 0 0
  - $pila2$= 3 4 -2 7 0 0
- Se extrae un valor
  - Se obtiene 7
  - $pila1$= 2 -2 4 0 0 0
  - $pila2$= 2 4 -2 0 0 0
- Se extrae un valor
  - Se obtiene -2
  - $pila1$= 1 4 0 0 0 0
  - $pila2$= 1 4 0 0 0 0
- Se extrae un valor
  - Se obtiene 4
  - $pila1$= 0 0 0 0 0 0
  - $pila2$= 0 0 0 0 0 0

Observaciones y conclusiones

- Con esta implementación se evita el uso de estructuras y se aprovecha la posición $[0]$ del arreglo de datos que originalmente no se utiliza.
- Los datos en el arreglo son inicializados con 0, esto hace difícil de ver cuando algunos de los datos ingresados han sido 0, sin embargo se puede saber cuales son datos "válidos" revisando el primer valor del arreglo el cual indica cuantos valores existen dentro del árbol. Otra alternativa podría ser representar los espacios vacíos del árbol con NAN o hacer un arreglo de tamaño variable de acuerdo a la cantidad de datos ingresados y extraídos.
- El ingresar los datos uno por uno a la $cola$ simulada puede resultar algo tedioso, sin embargo es la mejor manera de ir analizando y validando el comportamiento del mismo.

Problema 5

Para comparar las expresiones dadas se simplifica cada una de ellas

- $4^{log_2 n}\sqrt{n-3}$

$$4^{log_2 n}\sqrt{n-3}$$
$$(2^2)^{log_2 n}\sqrt{n-3}$$
$$2^{2log_2 n\sqrt{n-3}}$$
$$2^{log_2 n^2}\sqrt{n-3}$$
$$n^2\sqrt{n-3}$$
$$\sqrt{n^4}\sqrt{n-3}$$
$$\sqrt{n^4(n-3)}$$
$$\sqrt{n^5-3n^4}$$

Tomando el término más significativo dentro de la raíz cuadrada se concluye que:

$$4^{log_2 n}\sqrt{n-3} \leq O(\sqrt{n^5})$$

- $n + log(n)$

Para este caso solo se tiene que observar que el término más significativo es n, por tanto:

$$n + log(n) \leq O(n)$$

- $n^3 log_5 n$

Directamente la complejidad se obtiene como:

$$n^3 log_5 n \leq O(n^3 log(n))$$

- $\sum_{i=1}^{n} \frac{1}{i}$

Se define el $n - \acute{e}simo$ número armónico como la suma de los recíprocos de los primeros n número naturales. Para los números naturales se puede representar como:

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = \sum_{k=0}^{n-1} \int_0^1 x^k dx$$

H_n crece igual de rápido que el logaritmo natural de n. La razón es que la suma está aproximada por la integral: $\int_1^n \frac{1}{x} dx$ cuyo valor es $log(n)$. Tenemos entonces el siguiente limite:

$$\lim_{n \to \infty} H_n - log(n) = \gamma$$
$$\lim_{n \to \infty} H_n = \gamma + log(n)$$

Donde γ es un valor constante. Entonces se puede concluir:

$$\sum_{i=1}^{n} \frac{1}{i} \leq O(log(n))$$

- $n!$

Para esta expresion no hay mucho qué decir. Se obtiene directamente:

$$n! \leq O(n!)$$

- $n^3 |cos(n)|$

$|cos(n)|$ es un valor que va desde 0 hasta 1 por lo que el factor de mayor importancia en la expresión es n^3. Se concluye entonces que:

$$n^3 |cos(n)| \leq O(n^3)$$

Ordenamos las expresiones de $O()$

$$O(log(n)) \leq O(n) \leq O(\sqrt{n^5}) \leq O(n^3) \leq O(n^3 log(n)) \leq O(n!)$$

Por tanto:

$$\sum_{i=1}^{n} \frac{1}{i} \leq n + log(n) \leq 4^{log_2 n} \sqrt{n-3} \leq n^3 |cos(n)| \leq n^3 log_5 n \leq n!$$