Ayudantía 4

Carlos Lagos - carlos.lagosc@usm.cl

Mejor Caso, Peor Caso y Caso Promedio

- **Peor caso**: La situación más desfavorable para el algoritmo, donde el tiempo de ejecución es máximo.
- **Mejor caso**: La situación más favorable para el algoritmo, donde el tiempo de ejecución es mínimo.
- **Caso promedio**: El rendimiento esperado considerando una distribución típica de entradas.

Análisis Asintótico

Ejercicio 1

Algorithm 3 *BuscarOrdenado*$(A[i], x)$

```
1:  i ← √n
2:  while A[i] ◁ x do
3:      i ← i + √n
4:  end while
5:  d ← i
6:  i ← i − √n + 1
7:  while i ≥ d do
8:      if x ⊴ A[i] then
9:          break
10:     end if
11:     i ← i + 1
12: end while
13: if i ≤ d and x = a[i] then
14:     return i
15: else
16:     return -1
17: end if
```

1. ¿Cuál es el mejor caso?
2. ¿Cuál es el peor caso?
3. ¿Cuál es el caso promedio de éxito?
4. ¿Cuál es el caso promedio de fracaso?
5. ¿Cuál es el caso promedio general?

Ejercicio 2

Queremos realizar Q consultas sobre un arreglo de tamaño N. Cada consulta está definida por dos enteros L y R, y para cada consulta debemos obtener el valor máximo en el rango de índices desde L hasta R. Se propone una solución que divide el arreglo en bloques de tamaño B. Para cada bloque, se precalcula el valor máximo de los elementos que contiene. Al responder una consulta que abarca el rango de índices L a R, se aprovechan los bloques precalculados para obtener el valor máximo de manera eficiente en los bloques completamente incluidos en el rango. Para los elementos que están fuera de estos bloques y que caen dentro del rango L a R, se calcula el valor máximo de forma normal, es decir, examinando cada elemento individualmente.

- Calcula el peor caso del tiempo de ejecución para este algoritmo.
- Determina el tamaño óptimo B para minimizar el tiempo en el peor caso.

Análisis Amortizado

Ejercicio 1

Se ejecuta una secuencia de n operaciones sobre una estructura de datos. La i-ésima operación tiene un costo de i si i es una potencia de 2, y de 1 en caso contrario. En este contexto, se busca determinar el costo amortizado por operación.

Análisis Amortizado

Ejercicio 2

Se realizan una serie de operaciones sobre una pila cuyo tamaño nunca excede los k elementos. Después de k operaciones, se realiza un respaldo de la pila, creando una copia completa de sus elementos. Demuestre que el costo de n operaciones, incluida la copia de la pila, es $O(1)$ amortizado por operación.

Dividir y Conquistar

Ejercicio 1

Una fábrica tiene n máquinas que pueden ser utilizadas para fabricar productos. Tu objetivo es producir un total de t productos. Para cada máquina, conoces la cantidad de segundos que necesita para fabricar un solo producto. Las máquinas pueden trabajar simultáneamente, y puedes decidir libremente su programación.

¿Cuál es el tiempo mínimo necesario para fabricar t productos?

Debes resolver este problema utilizando búsqueda binaria, con una complejidad esperada de $O(n \log(A_{\max} \cdot t))$, siendo A el arreglo de los tiempos y $A_{\max}$ el valor máximo en A.

Dividir y Conquistar

Ejercicio 2

Sea $A[1..n]$ un arreglo de n números enteros. Un par $(A[i], A[j])$, con $1 \leq i, j \leq n$, es una inversión si $i < j$ y $A[i] > A[j]$. Usa la técnica dividir y conquistar para diseñar un algoritmo que cuente el número de inversiones en un arreglo en tiempo $\Theta(n \log n)$. Escribe la solución usando pseudocódigo (o algún lenguaje de programación).

Correctitud

Ejercicio 1

Dado un algoritmo para calcular la suma de los primeros `n` números naturales, verifica la correctitud del algoritmo.

```
int suma(int n) {
    int resultado = 0;
    int i = 1;
    while (i <= n) {
        resultado = resultado + i;
        i = i + 1;
    }
    return resultado;
}
```

Correctitud

Ejercicio 2

Demuestra que la función que calcula el factorial de un número entero positivo n es correcta.

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Correctitud

Ejercicio 3

Demuestra que la función que calcula el n-ésimo número de Fibonacci usando recursión es correcta usando inducción matemática.

```
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

Correctitud

Ejercicio 4

Dado un algoritmo de Bubble Sort para ordenar una lista de números enteros en orden ascendente, verifica la correctitud del algoritmo usando un invariante de ciclo.

FIN