Análisis de algoritmos

Deyner Navarro Badilla - 2020113009

Algoritmo de la tableta de chocolate:

```
int costo(int N, int M, int x[], int y[]) {

    if (N == 1 && M == 1)
        return 0;

    int mayorX = elementoMayor_Indice(x, N-1);      Tiempo: C1
    int mayorY = elementoMayor_Indice(y, M-1);
    int coste1 = 0;
    int coste2 = 0;
                                                            Tiempo: C2
    if ((mayorY == -1) || ((mayorX >= 0 && mayorY >= 0) && (x[mayorX] > y[mayorY]))) {

        coste1 = costo(mayorX + 1, M, x, y);               Tiempo: f(n/2)
        coste2 = costo(N - (mayorX + 1), M, x + (mayorX + 1), y);  Tiempo: f(n/2)

        return coste1 + coste2 + x[mayorX];  Tiempo: C3
    }
    else {
        coste1 = costo(N, mayorY + 1, x, y);
        coste2 = costo(N, M - (mayorY + 1), x, y + (mayorY + 1));

        return coste1 + coste2 + y[mayorY];
    }
}
```

Para hallar f(n):

$f(n) = C_1 + C_2 + f(n/2) + f(n/2) + C_3$

$f(n) = 2f(n/2) + C$

Para hallar O(n):

- Utilizar Inducción matemática

Asumo:

$f(n) = O(n \cdot \log_2 n)$

$f(m) \leq c \cdot (m \cdot \log_2(m)) \quad Por\ definicion\ de\ O\big(g(n)\big) = c \cdot g(n)$

Tomo un m cualquier que cumpla $m \leq n$

$m = \frac{n}{2}$

$f\left(\frac{n}{2}\right) \leq c \cdot \left(\frac{n}{2} \cdot \log_2\left(\frac{n}{2}\right)\right)$

Demostración: Sustitución en $f(n) = 2f\left(\frac{n}{2}\right) + c$

$$f(n) \leq \left(2 \cdot c \cdot \left(\frac{n}{2} \cdot \log_2\left(\frac{n}{2}\right)\right)\right) + c$$

$$f(n) \leq \left(c \cdot n \cdot \log_2\left(\frac{n}{2}\right)\right) + c$$

$$f(n) \leq (c \cdot n \cdot (\log_2 n - \log_2 2)) + c$$

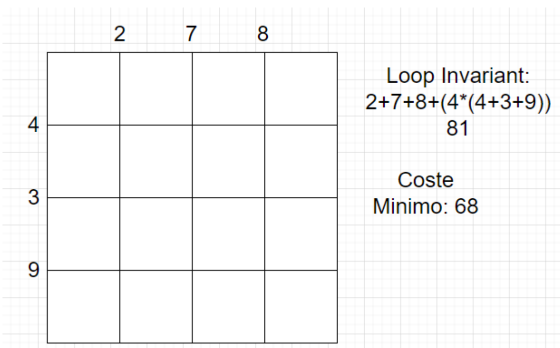
$$f(n) \leq (c \cdot n \cdot (\log_2 n - 1)) + c$$

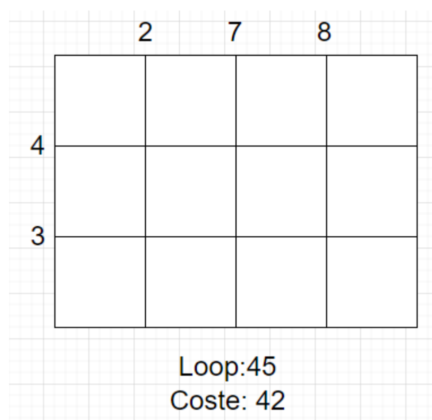
$$f(n) \leq cn\log_2 n - cn + c$$

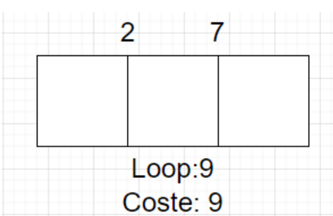
$$\therefore f(n) \leq cn\log_2 n - cn + c \leq cn\log_2 n \rightarrow f(n) \leq O(n\log_2 n)$$

Loop Invariant:

$$costeMinimo \leq (x1 + x2 + \ldots + xn) + (N * (y1 + y2 + \ldots + ym))$$







Algoritmo de Radixsort:

```
void radixSort(int *arr, int n, int max)
{
    int i;
    int j;
    int m;
    int p = 1;
    int index;
    int temp;
    int count = 0;

    list<int> pocket[10]; // 1

    for (i = 0; i < max; i++) {

        m = pow(10, i + 1);
        p = pow(10, i);

        for (j = 0; j < n; j++) {
            temp = arr[j] % m;
            index = temp / p;

            pocket[index].push_back(arr[j]);
        }
        count = 0;

        for (j = 0; j < 10; j++) {
            while (!pocket[j].empty()) {
                arr[count] = *(pocket[j].begin());
                pocket[j].erase(pocket[j].begin());
                count++;
            }
        }
    }
}
```

Se repite n veces por cada m

Se repite m veces

Se repite 10 veces

$$f(n) = 25mn + 1800m + 9$$

$$f(n) = O(mn)$$