Complejidad algoritmica

03 de Abril, 2019

Complejidad

Que es la complejidad algoritmica?

La complejidad algoritmica es la cantidad de operaciones que realiza un programa. El tiempo que tarda en ejecutarse esta directamente relacionado con la cantidad de operaciones que realiza.
Una computadora normalmente hace unas 10^8 operaciones por segundo.

Problema I

Consigna

Dado un número n par, devolver la mitad de n.

```
1   int n;
    cin >> n;
3   cout << n/2;
```

Como escala?

¿Qué pasa si n es 10?
¿Si n es 10^6?
¿Si n es 10^{18}?

Problema II

Consigna

Decir cual es el número más grande en un arreglo de números positivos
Sea el arreglo a[n]

```
1   int max = -INFINITO;
    for(int i = 0; i < n; i++)
3     if (a[i] > max)
        max = a[i];
5   cout << max << endl;
```

Como escala?

¿Qué pasa si n es 10?
¿Si n es 10^6?
¿Si n es 10^{18}?

¿Cómo calcular el orden?

```
1   int n, tmp, res = -1;
    cin  >> n;
3   for(i = 0; i < n; i++) {
      cin >> tmp;
5     res = max(res,tmp);
      cout << max << endl;
7   }
```

¿Cómo calcular el orden?

```
1   int n, tmp, res = -1;
    cin >> n;
3   for(i = 0; i < n; i++) {
      cin >> tmp;
5     res = max(res,tmp);
      cout << max << endl;
7   }
```

Operaciones

3*n + 4 operaciones

O(n)

¿Cómo calcular el orden?

```
int i, j, n;
cin >> n;
int a[n][n];
for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
        cin >> a[i][j];
        cout << "exito" << endl;
  }
}
for(i = 0; i < n; i++)
    cout << a[i][0] << endl;
```

¿Cómo calcular el orden?

```
1  int i, j, n;
   cin >> n;
3  int a[n][n];
   for(i = 0; i < n; i++) {
5      for(j = 0; j < n; j++) {
           cin >> a[i][j];
7          cout << "exito" << endl;
       }
9  }
   for(i = 0; i < n; i++)
11     cout << a[i][0] << endl;
```

Operaciones

$5 + 2 * n^2 +$ n operaciones

$O(n^2)$

Problema III

Enunciado

Te dan un arreglo A de tamaño n, y q consultas. Para cada consulta, te dan un número entero x y tenes que decir si x está en el arreglo o no.

Input

La primera linea tiene dos enteros, n y q ($1 \leq n,q \leq 10^5$). En la segunda línea hay n números separados por espacios, los elementos ai del arreglo. Luego siguen q líneas con un entero x en cada línea ($1 \leq ai,x \leq 10^6$)

Output

Para cada consulta, imprime en una nueva línea "YES" si pertenece y "NO" en caso contrario.

Problema III - Solución 1

```
1
int i,n,q,x;
3 cin >> n >> q;
for(i = 0; i < n; i++)
5   cin >> a[i];
while(q--) {
7     cin >> x;
      for(i = 0; i < n; i++) {
9       if (a[i] == x) {
          cout << "YES" << endl;
11        break;
        }
13    }
      if (i == n) {
15      cout << "NO" << endl;
      }
17 }
```

Time Limit Exceeded

¿Por qué?

- Una computadora juez puede hacer hasta 10^8 operaciones por segundo.
- Normalmente nos dan 1 segundo para resolver el problema
- Nuestro algoritmo en el peor caso hace 10^{10} operaciones $\rightarrow$ Nuestro algoritmo puede llegar a tardar 100 segundos

Búsqueda Binaria: buscando el 1000

Arreglo

[1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000] (16 elementos)

Búsqueda lineal

1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 16 consultas.

Búsqueda Binaria: buscando el 1000

Arreglo

[1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000] (16 elementos)

Búsqueda lineal

1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 16 consultas.

Búsqueda Binaria

- 1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000

Búsqueda Binaria: buscando el 1000

Arreglo

[1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000] (16 elementos)

Búsqueda lineal

1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 16 consultas.

Búsqueda Binaria

- 1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 42?

Búsqueda Binaria: buscando el 1000

Arreglo

[1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000] (16 elementos)

Búsqueda lineal

1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 16 consultas.

Búsqueda Binaria

- 1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 42?
- 43,46,51,55,70,100,1000

Búsqueda Binaria: buscando el 1000

Arreglo

[1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000] (16 elementos)

Búsqueda lineal

1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 16 consultas.

Búsqueda Binaria

- 1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 42?
- 43,46,51,55,70,100,1000 $\rightarrow$ 55?

Búsqueda Binaria: buscando el 1000

Arreglo

[1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000] (16 elementos)

Búsqueda lineal

1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 16 consultas.

Búsqueda Binaria

- 1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 42?
- 43,46,51,55,70,100,1000 $\rightarrow$ 55?
- 70,100,1000 $\rightarrow$

Búsqueda Binaria: buscando el 1000

Arreglo

[1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000] (16 elementos)

Búsqueda lineal

1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 16 consultas.

Búsqueda Binaria

- 1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 42?
- 43,46,51,55,70,100,1000 $\rightarrow$ 55?
- 70,100,1000 $\rightarrow$ 100?

Búsqueda Binaria: buscando el 1000

Arreglo

[1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000] (16 elementos)

Búsqueda lineal

1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 16 consultas.

Búsqueda Binaria

- 1,3,6,8,10,15,24,30,42,43,46,51,55,70,100,1000 $\rightarrow$ 42?
- 43,46,51,55,70,100,1000 $\rightarrow$ 55?
- 70,100,1000 $\rightarrow$ 100?
- 1000 $\rightarrow$ 4 consultas

Ganancia

En particular, en búsqueda binaria siempre divido en 2 el intervalo, entonces en el peor caso hago log2(n)+1 consultas.
En una lista de tamaño 16 (2^4) hago 5 consultas con búsqueda binaria y 16 con búsqueda lineal
En una lista de tamaño 1.073.741.824(2^{30}) son 31 consultas contra más de un millón.
Cuanto más grande el n más grande la ganancia

Errores frecuentes

Error número 1:

Búsqueda binaria funciona en una lista ordenada
Si van a ordenar, usen el sort de c++.
Complejidad de sort: n*log2(n)

Error número 2:

Se me clava el programa!
Fijensen bien los índices, si se equivocan puede terminar en un loop infinito
Usen cout para ver en qué parte se clava

Sintáxis de sort

Sort para un arreglo

```
int a[MAXN];
sort(a, a+n);
```

Sort para un vector

```
vector<int> vect;
sort(vect.begin(), vect.end());
```

Si lo quieren de mayor a menor

```
reverse(a, a+n);
reverse(vect.begin(), vect.end());
```

Volviendo al ejercicio

enunciado

Te dan un arreglo A de tamaño n, y q consultas. Para cada consulta, te dan un número entero x y tenes que decir si x está en el arreglo o no.

Input

La primera linea tiene dos enteros, n y q ($1 \leq n,q \leq 10^5$). En la segunda línea hay n números separados por espacios, los ai elementos del arreglo. Luego siguen q líneas con un entero x en cada línea ($1 \leq ai,x \leq 10^5$)

Output

Para cada consulta, imprime en una nueva línea "YES" si pertenece y "NO" en caso contrario.

Problema III - Solución 2 (parte 1)

Idea

Ordenar el arreglo, y hacer una busqueda binaria para cada query.

Problema III - Solución 2 (parte 1)

Idea

Ordenar el arreglo, y hacer una busqueda binaria para cada query.

```
#include <iostream>
#include <algorithm>
using namespace std;
int a[100000];
int main() {
  ios::sync_with_stdio(false);
  int i,n,q,x,l,r,m;
  cin >> n >> q;
  for(i = 0; i < n; i++)
    cin >> a[i];
  sort(a,a+n);
```

Problema III - Solución 2 (parte 2)

```
    while(q--) {
      cin >> x;
      l = 0, r = n-1;
      while(l <= r) {
        m = (l + r)/2;
        if (a[m] == x) {
          cout << "YES" << endl;
          break;
        } else if (a[m] < x){
          l = m+1;
        } else {
          r = m-1;
        }
      }
      if (l > r)
        cout << "NO" << endl;
    }
}

```

Ganancia

Como la complejidad del sort es $n * log(n)$, la de búsqueda binaria es $log(n)$, y repetimos esto q veces, la complejidad es $O(n * log(n) + n * log(n)) = O(2 * n * log(n)) = O(n * log(n))$

Problema III - Idea 3

Para cada entero x del input, marco como True en un arreglo de booleanos, en la posicion x.

Poblema III - Solucion 3

```
int n, q, x, res;
bool a[100000];

for(int i = 0; i < n; i++) {
    cin >> x;
    a[x] = true;
}
cin >> q;
while(q--) {
    cin >> x;
    if (a[x]) {
        cout << "YES" << endl;
    } else {
        cout << "NO" << endl;
    }
}

```

Ganancia

Como acceder a un arreglo es O(1), y procesar las querys es O(n), la complejidad queda O(n), que es la mejor de las 3

Proglema IV

Problema

Te dan un arreglo de enteros a de tamaño n, y q consultas. Para cada consulta, te dan dos números enteros l y r, y tenes responder de a[l] + a[l+1] + .. + a[r].

Input

La primera linea tiene dos enteros, n y q ($1 \leq n,q \leq 10^5$). En la segunda línea hay n números separados por espacios, los elementos ai del arreglo. Luego siguen q líneas con dos enteros l,r en cada línea ($1 \leq l,r \leq 10^5$)

Output

Para cada consulta, imprime $\sum_{i=l}^{r} a[i]$

Ejemplo

Ejemplo

$l \rightarrow 3$

$r \rightarrow 6$

A[8] = {-1, 2, 4, -3, 5 ,2, -5, 2};

Ejemplo

Ejemplo

l $\rightarrow$ 3

r $\rightarrow$ 6

A[8] = {-1, 2, 4, -3, 5 ,2, -5, 2};

Respuesta $\rightarrow$ [4, -3, 5 ,2] = 8

Problema IV - Idea 1

Para cada query, recorrer el arreglo de l, a r, e ir sumando en una variable temporal.

Problema IV - Solución 1

```
1
int i,n,q,l,r, a[100000];
3 cin >> n >> q;
for(i = 0; i < n; i++)
5   cin >> a[i];
for(int k = 0; k < q; k++) {
7     cin >> l >> r;
      l--;
9     r--;
      int aux = 0;
11    for(i = l; i <= r; i++) {
          aux += a[i];
13    }
      cout << aux << endl;
15 }
```

Time Limit Exceeded

¿Por qué?

- Nuestro algoritmo en el peor caso hace 10^{10} operaciones $\rightarrow$ Nuestro algoritmo puede llegar a tardar 100 segundos

Sumas parciales

Dado un arreglo de números $a_0 ... a_{n-1}$. Definimos $sum(l, r) =$ "suma del sub-arreglo $a_l ... a_r$".

- Observemos que $sum(l, r) = sum(0, r) - sum(0, l - 1)$
- Luego, si precalculamos $sum(0, i)$ para todo i, podemos responder cada pregunta en $O(1)$.
- La recurrencia es simple:
  - $sum(0, 0) = 0$
  - $sum(0, i + 1) = sum(0, i) + a_i$.

Sumas parciales - Código

```
int a[MAXN],sp[MAXN+1];int n;
void init_sum(){
  sp[0] = a[i];
  for(int i = 1; i < n; ++i)
    sp[i] = sp[i-1] + a[i];
}
int sum(int l, int r){
  return sp[r] - sp[l-1];
}
```

La idea de hacer sumas parciales aparece en muchos problemas, es bueno tenerla presente.

Volviendo al ejercicio

Problema

Te dan un arreglo de enteros a de tamaño n, y q consultas. Para cada consulta, te dan dos números enteros l y r, y tenes responder de a[l] + a[l+1] + .. + a[r].

Input

La primera linea tiene dos enteros, n y q ($1 \leq n,q \leq 10^5$). En la segunda línea hay n números separados por espacios, los elementos ai del arreglo. Luego siguen q líneas con dos enteros l,r en cada línea ($1 \leq l,r \leq 10^5$)

Output

Para cada consulta, imprime $\sum_{i=l}^{r} a[i]$

Problema IV - Idea 2

Precalcular las sumas parciales de 0 a n-1, y responder las querys en O(1).

Problema IV - Solución 2

```
1
int i,n,q,l,r, a[100001], s[100001];
3 cin >> n >> q;

5 for(i = 1; i <= n; i++)
    cin >> a[i];
7
for(i = 1; i <= n; i++) {
9     s[i] = s[i-1] + a[i];
}
11
while(q--) {
13     cin >> l >> r;
       cout << s[r] - s[l-1] << endl;
15 }
```

Input

Prestar atención

Los datos del input te limitan de qué orden puede ser tu algoritmo.
Muchas veces conviene leer el tamaño del input antes de empezar a pensar en una solución

Tamaño de Input	Complejidad máxima
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n\log(n))$ o $O(n)$
$n > 10^6$	$O(1)$ o $O(\log(n))$

Si no les queda muy claro qué complejidad tiene algún algoritmo o programa y por qué, no duden en consultarnos

Ejercicios con búsqueda binaria

http://codeforces.com/problemset/problem/600/B
http://www.spoj.com/problems/AGGRCOW/
https://www.hackerearth.com/practice/algorithms/searching/binary-search/practice-problems/algorithm/bishu-and-soldiers/
http://codeforces.com/problemset/problem/348/A
http://codeforces.com/problemset/problem/743/B