Selection

Selection (Search for the k-th smallest element)

Problem: Given an array of n elements, the selection problem is to find the k-th smallest element.

Direct solution: Sort the n items and access the k-th smallest one. Cost: $O(n \log n)$.

D&C solution: Is it possible to find a more efficient algorithm? Using the idea of the partition algorithm:

-**Divide (partition):** The array $A[p..r]$ is partitioned (reorganized) in two subvectors $A[p..q]$ and $A[q+1..r]$, so that the elements of $A[p..q]$ are less than or equal to the pivot and those of $A[q+1..r]$ are greater or equal.

-**Conquer:** We search in the corresponding subvector doing recursive calls to the algorithm.

-**Combine:** If $k \leq q$, then the k-th smallest item will be in $A[p..q]$. If not, it will be in $A[q+1..r]$.

Recursive algorithm for the Selection problem

```
public static <T extends Comparable<T>>
T seleccion(T v[], int k) {
  return seleccion(v,0,v.length-1,k-1);
}
private static <T extends Comparable<T>>
T seleccion(T v[], int p, int r, int k) {
  if (p == r)
    return v[k];
  else {
    int q = particion(v, p, r);
    if (k <= q)
      return seleccion(v, p, q, k);
    else
      return seleccion(v, q+1, r, k);
  }
}
```

Iterative algorithm for the Selection problem

With *tail recursion*:

```
private static <T extends Comparable<T>>
T seleccion(T v[], int p, int r, int k) {
  while (p < r) {
    int q = particion(v, p, r);
    if (k <= q)
      r = q;
    else
      p = q + 1;
  }
  return v[p];
}
```

Exercise: Trace for the recursive and iterative algorithm for Selection with $A = \{31, 23, 90, 0, 77, 52, 49, 87, 60, 15\}$ y $k = 7$.

Analysis of the Selection algorithm

Worst case: Ordered vector in a non-decreasing way and we look for the major element ($k = n$). Cost: $O(n^2)$.

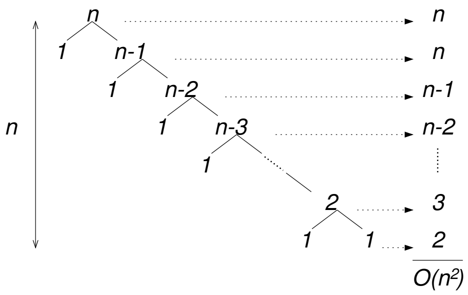

Figura : Worst case for selection

$$T(n) = \begin{cases} c, & \text{if } n \leq 1; \\ T(n-1) + c'n, & \text{if } n > 1 \end{cases}$$

Analysis of the Selection algorithm

Best case: The item to look for is the minor ($k = 1$) or the biggest ($k = n$) and this acts as a pivot $\rightarrow$ A single call to partition. Cost: $O(n)$.

Average case: with certain assumptions of randomness, the Selection algorithm has a cost $\in \Theta(n)$ (page 188, Cormen 90), (page 167, Horowitz 98).

Suppose that in each call to the partition algorithm, the problem is reduced to half. Recurrence equation:

$$T(n) = \begin{cases} c, & \text{if } n \leq 1; \\ T(n/2) + c'n, & \text{if } n > 1 \end{cases}$$

Analysis of the Selection algorithm

$$T(n) = \begin{cases} c, & \text{if } n \leq 1; \\ T(n/2) + c'n, & \text{if } n > 1 \end{cases}$$

$$\begin{aligned}
T(n) &= T(n/2) + c'n \\
&= (T(n/2^2) + c'n/2) + c'n = T(n/2^2) + (n + n/2)c' \\
&= (T(n/2^3) + c'n/2^2) + (n + n/2)c' = T(n/2^3) + n(1 + 1/2 + 1/2^2)c' \\
&\dots \\
&= T(n/2^i) + c'n\sum_{j=0}^{i-1} 1/2^j \\
&\quad //\text{Serie geométrica: } \sum_{j=0}^{i} x^i = \frac{x^{i+1}-1}{x-1}; // \sum_{j=0}^{i-1} 1/2^j = \frac{1/2^i - 1}{1/2 - 1} = \frac{2(2^i-1)}{2^i} \\
&= T(n/2^i) + n\frac{2(2^i-1)}{2^i}c' \qquad \{n/2^i = 1, i = \log n\} \\
&= T(1) + n\frac{2(n-1)}{n}c' = c_1 + 2(n-1)c' \in O(n)
\end{aligned}$$

Other problems

Binary search

Given a vector of size n, ordered in increasing order, find x.

- A sequential search has cost $O(n)$
- D&C: Taking advantage of the fact that the vector is ordered $O(\log n)$.

```
/** 0<=inicio<=fin<v.length */
public static <T extends Comparable<T>>
int busBinaria (T[] v, int inicio, int fin, T x) {
  if (inicio>fin) return -1;
  int mitad = (inicio+fin)/2;
  int cmp   =  v[mitad].compareTo(x);
  if (cmp == 0) return mitad;
  if (cmp >  0) return busBinariaRec(v,inicio,mitad-1,x);
  else          return busBinariaRec(v,mitad+1,fin,x);
}
```

- Sometimes it is only necessary to solve a subproblem, some authors name this case *reduce and conquer*

Binary search

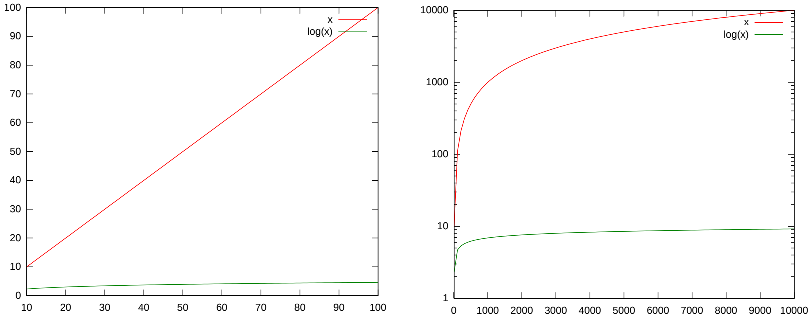

Figura : Linear and logarithmic cost comparison

Computation of the integer power

We want to calculate the power a^n where n is a non negative integer and being a a value with the associative property.

An iterative scheme $a^n = a \times a \times \cdots \times a$ has a cost $\Theta(n)$.

Taking advantage of the property $a^n \times a^m = a^{(n+m)}$ we can do:

```
double elevar(double a, int n) {
  if (n==0) return 1;
  if (n==1) return a; // no hace falta
  double aux = elevar(a,n/2);
  if (n%2==0)
    return aux*aux;
  return aux*aux*a;
}
```

$$T(n) = \begin{cases} k, & \text{if } n \leq 1; \\ T(n/2) + k', & \text{if } n > 1 \end{cases}$$

Cost $\Theta(\log n)$

Temporal complexity

Temporary complexity D&C: dividing recurrence

Let's solve a problem T for an instance of size n using D&C:

-**Divide** the original problem of size n in a subproblems of size n/b. Let the cost of this division be $D(n)$.

-**Conquer** the a subproblems of size n/b. This cost is $aT(n/b)$.

-**Combine** the subproblems for the originl problema. Let this cost $C(n)$.

Example: mergesort

- **Divide** is to calculate the average index of the vector. Cost $\Theta(1)$.
- **Conquer** in 2 subproblems of size $n/2$.
- **Combine** is the merge operation with cost $\Theta(n)$.

Temporary complexity D&C: dividing recurrence

General equation of dividing recurrence:

$$T(n) = \begin{cases} c, & \text{if } n \leq n_0; \\ aT(n/b) + D(n) + C(n), & \text{if } n > n_0 \end{cases}$$

When $D(n) + C(n)$ is $\Theta(n^k)$ we have this particular case:

$$T(n) = \begin{cases} c, & \text{if } n \leq n_0; \\ aT(n/b) + \Theta(n^k), & \text{if } n > n_0 \end{cases}$$

being n_0 the size below which we apply the base case of the recursion.

Temporal complexity D&C: subtractive recurrence

Recursive calls from a size n are to subproblems of size $n - c$, which corresponds to this recurrence equation:

$$T(n) = \begin{cases} k, & \text{if } n \leq n_0; \\ aT(n-c) + g(n), & \text{if } n > n_0 \end{cases}$$

Examples:

- Unbalanced Mergesort (MergesortBad) had cost $\Theta(n^2)$.

Temporal complexity D&C: master theorems

So far we have calculated the cost of a recursive algorithm using the *substitution method*, let's look at a couple of master theorems very useful for most of typical cases.

Master theorem for dividing recurrence: The solution to the equation $T(n) = aT(n/b) + \Theta(n^k)$, with $a \geq 1$ and $b > 1$, is:

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > b^k \\ O(n^k \log n) & \text{if } a = b^k \\ O(n^k) & \text{if } a < b^k \end{cases}$$

Temporal complexity D&C: master theorems

Example(*selection algorithm*) $a = 1, b = 2, k = 1$.
$T(n) \in O(n)$

$$T(n) = \begin{cases} 1, & \text{if } n \leq 1; \\ T(n/2) + n, & \text{if } n > 1 \end{cases}$$

Example(*mergesort, quicksort algorithms*) $a = 2, b = 2, k = 1$.
$T(n) \in O(n \log n)$

$$T(n) = \begin{cases} 1, & \text{if } n \leq 1; \\ 2T(n/2) + n, & \text{if } n > 1 \end{cases}$$

Temporal complexity D&C: master theorems

Master theorem for the subtracting recurrence: The solution to the equation

$$T(n) = \begin{cases} k, & \text{if } n \leq n_0; \\ aT(n-c) + \Theta(n^k), & \text{if } n > n_0 \end{cases}$$

has this cost:

$$T(n) = \begin{cases} \Theta(n^k) & \text{if } a < 1 \\ \Theta(n^{k+1}) & \text{if } a = 1 \\ \Theta(a^{n/c}) & \text{if } a > 1 \end{cases}$$