CS 4310 Algorithms: III. k-th Order Selection (Divide and Conquer Methods) and Recurrence Relations

E. de Doncker[1]

[1]Dept. of Computer Science, WMU, U.S.A.

Spring 2024

Outline

1. k-th Order Selection
   - Algorithm *Selection ( )*
   - Randomized (probabilistic) selection algorithm
   - Selection using the median of medians
2. Solving recurrences using the characteristic equation
   - Homogeneous linear recurrence
   - Nonhomogeneous linear recurrence

Outline

1. k-th Order Selection
   - Algorithm *Selection ( )*
   - Randomized (probabilistic) selection algorithm
   - Selection using the median of medians
2. Solving recurrences using the characteristic equation
   - Homogeneous linear recurrence
   - Nonhomogeneous linear recurrence

k-th order selection

Problem: Determine the k-the smallest key in a given array A
Section 8.5.4 [2], and Section 3.6 [1].
Algorithms make use of a *Partition ( )* function (as in *QuickSort ( )*).
Recursive algorithm Selection ( ): called as $Selection\,(1, n, k)$ on top level.
See alg. *selection ( )* in [2], or *Select1 ( )* (Program 3.17) in [1].

```
Algorithm Selection (low, high, k) {
    // Determine k-th smallest key in unsorted array A (range low to high)
    // returned as the function value

    if (low == high) return A[low];
    else {
        Partition (low, high, pivotindex);    // Partition array so that
                                  // all elements < A [pivotindex] are to its left
                                  // and all elements > A [pivotindex] are to its right
        if (k == pivotindex) return A [pivotindex];   // k-th smallest found
        else if (k < pivotindex) return Selection (low, pivotindex − 1, k);
        else return Selection (pivotindex + 1, high, k);
    }
}
```

Selection ( ) time complexity

The time is measured as the number of comparisons with the *pivotitem* in *Partition ( )*.

The worst case time $W(n)$ for an array of length n is incurred when *Partition ( )* splits off only the *pivotitem* in each recursive call, i.e., the array size decreases by 1 in each recursive call. A situation where this happens is when the array is sorted in nondecreasing order and $k = n$.
Thus the worst case time for *Selection ( )* is (as for *QuickSort ( )*):

$$W(n) = \frac{n(n-1)}{2} = \Theta(n^2)$$

The average case time $Av(n)$ is much better than $W(n)$,
Note that *Selection ( )* makes one recursive call on each level of the recursion.

Under the assumptions that all values of k are given with equal frequency, and all values of *pivotindex* occur with equal frequency, it can be shown [2] that

$$Av(n) = \Theta(n) \quad \text{(linear in } n\text{)}$$

This is done by solving a recurrence relation for $Av(n)$.

Randomized (probabilistic) selection algorithm

Algorithm *RSelection ( )* is as *Selection ( )*, but calls *RPartition ( )* (cf., *partition3 ( )* in [2]) where randomization is applied: *pivotitem* is set to $A[randspot]$, with *randspot* a random index in the range $[low, high]$ based on *random( )*, which draws from a uniform distribution.

```
Algorithm RPartition (low, high, pivotindex) {
    arraysize = high − low + 1;
    randspot = low + random( ) % arraysize;
    pivotitem = A [randspot];
    j = low;
    for ( i = low + 1; i ≤ high; i++ )
        if ( A[i] < pivotitem ) { j++; Exchange A[i] and A[j]; }
    pivotindex = j;
    Exchange A[low] and A[pivotindex];
}
```

It can be shown [2] that the expected value time complexity of randomized k-th order selection is $\mathcal{O}(n)$, independent of k (the expectation is over the space of the randomizer outputs).

Selection using the median of medians

We want to improve on the quadratic worst case of the selection algorithm that uses the *Partition ( )* function with *pivotitem* initialized at $A[low]$.

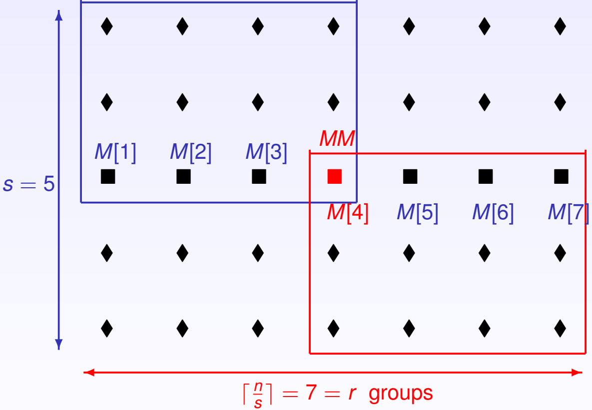
We would like to ensure that the array gets partitioned about in the middle (instead of at one end) at each recursive call. Therefore the *pivotitem* will be set to an element that is approximately the median of the array.
This element will be found by:
– dividing the array into groups (we consider $n/5$ groups of size 5, assuming n is an odd multiple of 5; other sizes are possible);
– take the median of each group;
– set pivotitem to the median of the array of medians.

The worst case time complexity of k-th order selection using the median of medians is $\mathcal{O}(n)$.

Selection using the median of medians

Example: $n = 35$, divide into $\lceil \frac{n}{s} \rceil = 7 = r$ groups of size $s = 5$

Median of k elements $= \lceil \frac{k}{2} \rceil th = \lfloor \frac{k+1}{2} \rfloor th$ *smallest* element ($3rd$ smallest for $k = 5$).



There are at least $\lceil \frac{s}{2} \rceil \lceil \frac{r}{2} \rceil = \lceil \frac{5}{2} \rceil \lceil \frac{7}{2} \rceil = 3 \times 4 = 12$ elements $\leq MM$ (blue rectangle)

and 12 elements $\geq MM$ (red rectangle).

Selection using the median of medians

Algorithm *MMSelection (A, low, high, k)* is as *Selection ( )*, but passes the array, and calls *MMPartition ( )* (cf., *selection2 ( )* and *partition2 ( )* in [2]).

Algorithm *MMPartition (A, low, high, pivotindex)* {
  $arraysize = high - low + 1;$
  $r = \lceil arraysize/5 \rceil;$ // number of groups
  for $(\, i = 1;\ i \le r;\ i{+}{+}\,)$ {   // determine median of each group
    $first = low + 5 * i - 5;$ // first index of group
    $last = \min\,(low + 5 * i - 1, arraysize);$ // last index of group
    $M[i] =$ median of $A[first]$ through $A[last]$;
  }
  $pivotitem =$ *MMSelection* $(M, 1, r, \lfloor (r+1)/2 \rfloor\,);$ // get approx. median of medians
  $j = low;$
  for $(\, i = low;\ i \le high;\ i{+}{+}\,)$ // partition around *pivotitem*
    if $(\, A[i] == pivotitem\,)$ { Exchange $A[i]$ and $A[j]$; $mark = j;\ j{+}{+};$ }
    else if $(\, A[i] < pivotitem\,)$ { Exchange $A[i]$ and $A[j]$; $j{+}{+};$ }
  $pivotindex = j - 1;$
  Exchange $A[mark]$ and $A[pivotindex]$;   // put *pivotitem* at *pivotindex*
}

Project ideas

– Multiplication of large integers
– Algorithms for computational geometry:
* Convex hull: compare algorithms for convex hull
* Line segments intersection problem: given n line segments, determine if any two line segments intersect.
* Closest pair problem: given n points, find a pair of points with the smallest distance between them.

Solving recurrences using the characteristic equation

Refer to Appendix B.2 [2]

Homogeneous linear recurrence

A *homogeneous linear recurrence* is a recurrence relation of the form

$$a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} = 0,$$

where k and the a_j coefficients are constants.
Its *characteristic equation* is

$$a_0 r^k + a_1 r^{k-1} + \ldots + a_{k-1} r + a_k = 0.$$

If the characteristic equation has k distinct roots, $r_1, r_2, \ldots, r_k$, i.e., it is factored as

$$(r - r_1)(r - r_2)\ldots(r - r_k) = 0,$$

then the recurrence relation is solved by

$$t_n = c_1 r_1^n + c_2 r_2^n + \ldots + c_k r_k^n,$$

where the c_j are arbitrary constants (that can be determined by initial conditions for the recurrence relation).

Homogeneous linear recurrence - Examples

Example ($k = 2$):

$$\begin{cases} t_n - 3t_{n-1} + 2t_{n-2} = 0 & n \geq 2 \\ t_0 = 3 \\ t_1 = 4 \end{cases}$$

Characteristic equation: $r^2 - 3r + 2 = 0$

Roots of the quadratic equation:

Discriminant $D = 3^2 - 4(1)(2) = 1$, $r_{1,2} = \frac{3 \pm \sqrt{D}}{2} = \frac{3 \pm \sqrt{1}}{2} = \begin{cases} r_1 = 4/2 = 2 \\ r_2 = 2/2 = 1 \end{cases}$

$\Longrightarrow \; t_n = c_1 r_1^n + c_2 r_2^n = c_1 2^n + c_2 1^n = c_1 2^n + c_2$

c_1 and c_2 can be determined from the initial conditions. Plug $n = 0$ and $n = 1$ into the solution for t_n: $\begin{cases} t_0 = c_1 2^0 + c_2 = c_1 + c_2 = 3 & (1) \\ t_1 = c_1 2^1 + c_2 = 2c_1 + c_2 = 4 & (2) \end{cases}$

This yields a linear system of two equations in the two unknowns c_1 and c_2, and solved by:

$(1) \Longrightarrow \; c_2 = 3 - c_1$

$(2) \Longrightarrow \; 2c_1 + c_2 = 2c_1 + 3 - c_1 = c_1 + 3 = 4$

$\Longrightarrow c_1 = 1$ and $c_2 = 3 - c_1 = 2 \Longrightarrow t_n = 2^n + 2$

Homogeneous linear recurrence - Examples

Now let's check if our solution is correct by pluggng it back into the original recurrence.
Check:
$t_n - 3t_{n-1} + 2t_{n-2} = (2^n + 2) - 3(2^{n-1} + 2) + 2(2^{n-2} + 2)$
$= 2^n - 3(2^{n-1}) + 2^{n-1} + 2 - 6 + 4 = 2^n - 2(2^{n-1}) = 0$
Thus the solution, $t_n = 2^n + 2$ indeed makes the right hand side of the recurrence evaluate to 0.

Example: simple divide and conquer matrix multiplicaton
We solve the following recurrence for the number of multiplications in multiplying two $n \times n$ matrices (where n is a power of 2).

$$\begin{cases} T(n) = 8T(\frac{n}{2}) & n \geq 2,\ n \text{ a power of 2} \\ T(1) = 1 \end{cases}$$

We set $n = 2^k$ and denote $t_k = T(2^k) = T(n)$. Then also, $T(\frac{n}{2}) = T(2^{k-1}) = t_{k-1}$.
That yields the recurrence

$$\begin{cases} t_k - 8t_{k-1} = 0 & k \geq 1 \\ t_0 = 1 \end{cases}$$

Homogeneous linear recurrence - Examples

Characteristic equation: $r - 8 = 0$, root $r = r_1 = 8$
$\Longrightarrow\ t_k = c_1 8^k = c_1(2^3)^k = c_1(2^k)^3 = c_1 n^3$
$\Longrightarrow\ T(n) = c_1 n^3,\ T(1) = c_1 = 1\ \Longrightarrow\ T(n) = n^3$

Check:
Plug $T(n) = n^3$ into the recurrence. The right hand side is
$8T(\frac{n}{2}) = 8(\frac{n}{2})^3 = n^3 = T(n)$ (= left hand side).

Example: Strassen's (divide and conquer) matrix multiplicaton
We solve the following recurrence for the number of multiplications in Strassen's matrix multication for two $n \times n$ matrices (where n is a power of 2).

$$\begin{cases} T(n) = 7T(\frac{n}{2}) & n \geq 2,\ n \text{ a power of } 2 \\ T(1) = 1 \end{cases}$$

We set $n = 2^k$ and denote $t_k = T(2^k) = T(n)$. Then also, $T(\frac{n}{2}) = T(2^{k-1}) = t_{k-1}$.

Homogeneous linear recurrence - Examples

That yields the recurrence

$$\begin{cases} t_k - 7t_{k-1} = 0 & k \geq 1 \\ t_0 = 1 \end{cases}$$

Characteristic equation: $r - 7 = 0$, root $r = r_1 = 7$

$\Longrightarrow \ t_k = c_1 7^k = c_1 (2^{\log_2 7})^k = c_1 (2^k)^{\log_2 7} = c_1 n^{\log_2 7}$

$\Longrightarrow T(n) = c_1 n^{\log_2 (7)}, \ T(1) = c_1 = 1 \Longrightarrow T(n) = n^{\log_2 7}$

Check:

Plug $T(n) = n^{\log_2 (7)}$ into the recurrence. The right hand side is

$7T(\frac{n}{2}) = 7(\frac{n}{2})^{\log_2 7} = 7\frac{n^{\log_2 7}}{2^{\log_2 7}} = n^{\log_2 7} = T(n)$ (= left hand side).

Root with multiplicity m

Refer to Appendix B.2 [2]

Homogeneous linear recurrence: root with multiplicity m

If the *characteristic equation*,

$$a_0 r^k + a_1 r^{k-1} + \ldots + a_{k-1} r + a_k = 0,$$

of the homogeneous linear recurrence has a root $r = r_j$ with multiplicity m, i.e., has a factor $(r - r_j)^m$, then this root leads to the following terms in the solution for t_n:

$$c_1 r_j^n + c_2 n\, r_j^n + c_3\, n^2 r_j^n + \ldots + c_m\, n^{m-1} r_j^n,$$

where the c_ℓ are arbitrary constants (that can be determined by initial conditions for the recurrence relation).

Nonhomogeneous linear recurrence

Nonhomogeneous linear recurrence

The recurrence is of the form

$$a_0 t_n + a_1 t_{n-1} + \ldots + a_k t_{n-k} = f(n),$$

where k and the a_j coefficients are constants, and $f(n)$ is a nonzero function.
For a right hand side function of the form

$$f(n) = p_1(n)\, b_1^n + \ldots + p_s(n)\, b_s^n$$

where each b_i is constant, and $p_i(n)$ is a polynomial in n of some degree d_i, $i = 1, 2, \ldots, s$, the *characteristic equation* is

$$(a_0 r^k + a_1 r^{k-1} + \ldots + a_{k-1} r + a_k)(r - b_1)^{d_1+1} \ldots (r - b_s)^{d_s+1} = 0,$$

and the solution t_n of the recurrence is derived from the roots of this equation.

Nonhomogeneous linear recurrence - Examples

Example: $W(n)$ for binary search

$$\begin{cases} W(n) = W(\frac{n}{2}) + 1 & n > 1,\ n \text{ a power of 2} \\ W(1) = 1 \end{cases}$$

Let $n = 2^k,\ t_k = W(2^k) = W(n)$, then the recurrence yields

$$\begin{cases} t_k - t_{k-1} = 1 = p_1(n)\, b_1^n \\ t_0 = 1 \end{cases}$$

The characteristic equation is: $(r-1)(r-b_1)^{d_1+1} = (r-1)(r-1) = (r-1)^2 = 0$, since $b_1 = 1$, and $p_1(n) = 1$ is a polynomial of degree $d_1 = 0$.

$\Longrightarrow\ t_k = c_1 1^k + c_2 k\, 1^k \ \Longrightarrow\ W(n) = c_1 + c_2 \log_2 n$

The coefficients c_1 and c_2 are determined using the initial conditions, i.e., by plugging $n = 1$ and $n = 2$ into the solution for $W(n)$.

Since $W(2)$ is not given, we calculate it from the recurrence: $W(2) = W(1) + 1 = 2$.

$n = 1: \quad W(1) = c_1 + c_2 \log_2 1 = c_1 = 1$

$n = 2: \quad W(2) = c_1 + c_2 \log_2 2 = c_1 + c_2 = 2 \Longrightarrow c_2 = 1$

Thus $W(n) = 1 + \log_2 n$.

Check: $W(1) = 1 + \log_2 1 = 1 + 0 = 1$;

(Right hand side of recurrence:) $W(\frac{n}{2}) + 1 = 1 + \log_2 \frac{n}{2} + 1 = 1 + \log_2 n = W(n)$

(= left hand side).

Nonhomogeneous linear recurrence - Examples

Example: Solve for the ($\mathcal{O}$) order of $T(n)$ using the characteristic equation:
structure $T(n) = 2T(\frac{n}{2}) + cn$, where c is a constant > 0 and n is a power of 2.
Set $n = 2^k$, $t_k = T(2^k) = T(n)$
$\implies t_k - 2t_{k-1} = c\,2^k$
The characteristic equation is: $(r-2)^2 = 0$
$\implies$ The solution is $t_k = c_1 2^k + c_2 k\, 2^k$
$\implies$ As a function of n: $T(n) = c_1 n + c_2 n \log_2 n = \mathcal{O}(n \log n)$

Example: $W(n)$ for merge sort

$$\begin{cases} W(n) = 2W(\frac{n}{2}) + n - 1 & n > 1,\ n \text{ a power of 2} \\ W(1) = 0 \end{cases}$$

Exercise: Solve using the method of the characteristic equation.
See Example B.19, Appendix B [2]

Bibliography

S. Sahni E. Horowitz and S. Rajasekeran.
Computer Algorithms/C++.
Computer Science Press, 2nd edition, 1998.
ISBN 0-7167-8315-0.

R. E. Neapolitan.
Foundations of Algorithms.
Jones and Bartlett, 5th edition, 2015.
ISBN 978-1-284-04919-0.