



We assume that are given a procedure MEDIAN that takes as parameters an array A and subarray indices p and r, and returns the value of the median element of A[p..r]A[p..r] in O(n) time in the worst case.

Given MEDIAN, here is a linear-time algorithm SELECT′ for finding the ith smallest element in A[p..r]. This algorithm uses the deterministic PARTITION algorithm that was modified to take an element to partition around as an input parameter.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | SELECT'(A, p, r, i)  if p == r  return A[p]  x = MEDIAN(A, p, r)  q = PARTITION(x)  k = q - p + 1  if i == k  return A[q]  else if i < k  return SELECT'(A, p, q - 1, i)  else return SELECT'(A, q + 1, r, i - k) |

Because x is the median of A[p..r], each of the subarrays A[p..q−1] and A[q+1..r] has at most half the number of elements of A[p..r]. The recurrence for the worst-case running time of SELECT′ is T(n)≤T(n/2)+O(n)=O(n).

The median can be obtained recursively as follows. Pick the median of the sorted array A. This is just O(1) time as median is the n/2th element in the sorted array. Now compare the median of A, call is a ∗ with median of B, b ∗ .

We have two cases.

• a ∗ < b∗ : In this case, the elements in B[n2..n] are also greater than a ∗ . So the median cannot lie in either A[1..n2] or B[n2..n]. So we can just throw these away and recursively solve a subproblem with A[n2..n] and B[1..n2].

• a ∗ > b∗ : In this case, we can still throw away B[1..n2] and also A[n2..n] and solve a smaller subproblem recursively. In either case, our subproblem size reduces by a factor of half and we spend only constant time to compare the medians of A and B. So the recurrence relation would be T(n) = T(n/2) + O(1) which has a solution T(n) = O(log n).

In either case, our subproblem size reduces by a factor of half and we spend only constant time to compare the medians of A and B. So the recurrence relation would be T(n) = T(n/2) + O(1) which has a solution T(n) = O(log n).