

$$\begin{aligned}
1. T(n) &= aT\left(\frac{n}{b}\right) + n^d \log^w n = a^2 T\left(\frac{n}{b^2}\right) + a\left(\frac{n}{b}\right)^d \log^w \frac{n}{b} + n^d \log^w n = \dots \\
&= \sum_{i=0}^{\log_b n} a^i \left(\left(\frac{n}{b^i}\right)^d \log^w \frac{n}{b^i}\right) < n^d \log^w n \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i \\
&\text{if } a < b^d, T(n) < n^d \log^w n \frac{1 - \left(\frac{a}{b^d}\right)^{\log_b n}}{1 - \frac{a}{b^d}} = O(n^d \log^w n) \\
&\text{if } a = b^d, T(n) = n^d \log^w n \log_b n = O(n^d \log^{w+1} n) \\
&\text{if } a > b^d, \text{ then } \log_b a > d, \\
&\quad \exists m > d, \log_b a > m, O(n^d \log^w n) < O(n^m), \\
&\quad \text{thus } T(n) = aT\left(\frac{n}{b}\right) + O(n^m) = O(n^{\log_b a})
\end{aligned}$$

2. Consider a Mom dividing by the number of k, k is odd.

We have $\frac{n}{k}$ groups, then there are $\frac{n}{k}$ medians. Choose the median of median v, which is no greater than $\frac{n}{2k}$ medians and no smaller than $\frac{n}{2k}$ medians. So v is no greater than $\frac{\frac{k+1}{2}n}{2k} = \frac{(k+1)n}{4k}$ numbers. Vice versa.

Now analyze the running time.

Select(S,k)

Pick a pivot v: $T\left(\frac{n}{k}\right) + O(n)$

Divide the array into 3 subsets: $O(n)$

L: $x < v$

M: $x = v$

R: $x > v$

if $k \leq L$, output Select(L,k); $T(L) \leq T\left(n - \frac{(k+1)n}{4k}\right)$

if $L < k \leq L+M$, output v;

if $L+M < k$, output Select(R,k-L-M); $T(R) \leq T\left(n - \frac{(k+1)n}{4k}\right)$

So, $T(n) \leq T\left(\frac{n}{k}\right) + T\left(\frac{(3k-1)n}{4k}\right) + O(n)$

When $k = 3$, $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$ the algorithm will require more recursive steps to find the median of each group.

When $k = 7$, $T(n) = T\left(\frac{n}{7}\right) + T\left(\frac{5n}{7}\right) + O(n)$ the overhead of finding the median of each group may also increase.

When $k = 9$, $T(n) = T\left(\frac{n}{9}\right) + T\left(\frac{13n}{18}\right) + O(n)$, outcome similar to $k = 7$.

3. Here's an algorithm that achieves this:

1) Compare the first two elements and let the larger one be the current maximum and the smaller one be the current minimum.

2) For each pair of consecutive elements starting from the third element, compare them and update the current maximum and minimum as follows:

a. If the first element is greater than the second element, compare the first element with the current maximum and the second element with the current minimum.

b. If the second element is greater than the first element, compare the second element with the current maximum and the first element with the current minimum.

After comparing all the pairs of elements, we have the maximum and minimum values.

Let's analyze the number of comparisons made by this algorithm. We compare the first two elements in one comparison. For each subsequent pair of elements, we make two comparisons. Therefore, we need a total of $2(n/2 - 1) = n - 2$ comparisons to compare all the pairs of elements.

We also need to compare the larger element of the first pair with the current maximum and the smaller element with the current minimum. We do this $n/2$ times. Therefore, we need $n/2$ comparisons for this step.

Thus, the total number of comparisons is $n - 2 + n/2 = 3n/2 - 2$

4. (a) We need h^2 bits to store the matrix. As we have w bits in one word, h is meant to be an integer smaller than \sqrt{w} .

(b) To compute the transpose of a one-word size binary matrix A , we can use a divide-and-conquer approach. We can split the matrix A into four smaller matrices, each of size $h/2 \times h/2$, as follows:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

where A_{11} , A_{12} , A_{21} , A_{22} are each $h/2 \times h/2$ matrices. We can then recursively compute the transpose of each of these smaller matrices, and combine them to obtain the transpose of the original matrix A .

To combine the transposes of the smaller matrices, we can use bitwise operations to interleave the bits of the transposed matrices. Specifically, we can use the following algorithm:

1. Check if the current matrix is 2×2 or 1×1 , if so, transpose it. If not, divide it into four matrices using the method above.
2. Merging: transpose the four sub-matrices, just swapping.

Time complexity: $T(w) = 4T(w/4) + O(1) = O(\log w)$

5. (a) Randomly select a vertex, if it is not with out-degree 0, let's consider the worst condition, that is today, to make the pass-way along the edges as long as possible. We can see that no matter which pattern the first vertex is, it comes to an end in no more than 4 steps, which is $O(1)$.

Algorithm:

1. Randomly select the first vertex, end if out-degree 0. Mark it to visited.
2. Recurse:
 1. Go along an out-edge to a next vertex. Mark it as visited. End if it's out-degree 0.
 2. Check the out-edges, end if point to a vertex visited.

(b)

Check the $n/2$ -th row, find the smallest number in the row. If the numbers above and beneath are both bigger than it, then it's a out-degree 0 vertex. Else, at least one of the two numbers are bigger than the chosen number. Delete the half of the matrix with the bigger number between the two numbers.

Do it recursively with column and row alternately, until we get a number which satisfies the statement.

The recursion takes $\sum_{i=0}^{\log n} \frac{n}{2}$ comparisons at most, which is $O(n)$.

(c)

Let H be the $n \times n$ grid graph described in the problem statement.

H can be created with a Hamiltonian circular, so that there won't be an out-degree 0 vertex.