1.

Groups of 7

The algorithm will work if the elements are divided in groups of 7. On each partitioning, the minimum number of elements that are less than (or greater than) x will be:

$$4\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{7}\right\rceil\right\rceil-2\right)\geq\frac{2n}{7}-8$$

The partitioning will reduce the subproblem to size at most $5n/7+8$. This yields the following recurrence:

$$T(n)=\begin{cases}\mathcal{O}(1) & \text{if } n<n_0\\ T(\lceil n/7\rceil)+T(5n/7+8)+\mathcal{O}(n) & \text{if } n\geq n_0\end{cases}$$

We guess $T(n)\leq cn$ and bound the non-recursive term with an:

$$\begin{aligned}T(n)&\leq c\lceil n/7\rceil+c(5n/7+8)+an\\&\leq cn/7+c+5cn/7+8c+an\\&=6cn/7+9c+an\\&=cn+(-cn/7+9c+an)\\&\leq cn\\&=\mathcal{O}(n)\end{aligned}$$

The last step holds when $(-cn/7+9c+an)\leq 0$. That is:

$$\begin{gathered}-cn/7+9c+an\leq 0\\\Downarrow\\c(n/7-9)\geq an\\\Downarrow\\\frac{c(n-63)}{7}\geq an\\\Downarrow\\c\geq\frac{7an}{n-63}\end{gathered}$$

By picking $n_0=126$ and $n\leq n_0$, we get that $n/(n-63)\leq 2$. Then we just need $c\geq 14a$.

Groups of 3

The algorithm will not work for groups of three. The number of elements that are less than (or greater than) the median-of-medians is:

$$2\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{3}\right\rceil\right\rceil-2\right)\geq\frac{n}{3}-4$$

The recurrence is thus:

$$T(n)=T(\lceil n/3\rceil)+T(2n/3+4)+\mathcal{O}(n)$$

We're going to prove that $T(n)=\omega(n)$ using the substitution method. We guess that $T(n)>cn$ and bound the non-recursive term with an.

$$\begin{aligned}T(n)&>c\lceil n/3\rceil+c(2n/3+2)+an\\&>cn/3+c+2cn/3+2c+an\\&=cn+3c+an && (c>0,a>0,n>0)\\&>cn\\&=\omega(n)\end{aligned}$$

The calculation above holds for any $c>0$.

2.

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  SELECT'(A, p, r, i)
2     if p == r
3        return A[p]
4     x = MEDIAN(A, p, r)
5     q = PARTITION(x)
6     k = q - p + 1
7     if i == k
8        return A[q]
9     else if i < k
10       return SELECT'(A, p, q - 1, i)
11    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 $T(n) \leq T(n/2)+O(n)=O(n)$.

3.

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).