5.4 Recurrences and Selection

One common problem that arises in algorithms is that of *selection*. In this problem you are given n distinct data items from some set which has an underlying order. That is, given any two items, it makes sense to talk about one being smaller than another. Given these n items, and some value i, $1 \leq i \leq n$, you wish to find the ith smallest item in the set. For example in the set $\{3, 1, 8, 6, 4, 11, 7\}$, the first smallest ($i = 1$) is 1, the third smallest ($i = 3$) is 4 and the seventh smallest ($i = n = 7$) is 11. An important special case is that of finding the *median*, which is the case of $i = \lceil n/2 \rceil$. Another important special case is finding percentiles; for example the 90th percentile is te case $i = \lceil .9n \rceil$. As this suggests, i is frequently given as some fraction of n.

Exercise 5.4-1 How do you find the minimum ($i = 1$) or maximum ($i = n$) in a set? What is the running time? How do you find the second smallest? Does this approach extend to finding the ith smallest? What is the running time?

Exercise 5.4-2 Give the fastest algorithm you can to find the median ($i = \lceil n/2 \rceil$).

In Exercise 5.4-1, the simple $O(n)$ algorithm of going through the list and keeping track of the minimum value seen so far will suffice to find the minimum. Similarly, if we want to find the second smallest, we can go through the list once, find the smallest, remove it and then find the smallest in the new list. This also takes $O(n + n - 1) = O(n)$ time. If we extend this to finding the ith smallest, the algorithm will take $O(in)$ time. Thus for finding the median, this method takes $O(n^2)$ time.

A better idea for finding the median is to first sort the items, and then take the item in position $n/2$. Since we can sort in $O(n \log n)$ time, this algorithm will take $O(n \log n)$ time. Thus if $i = O(\log n)$ we might want to run the algorithm of the previous paragraph, and otherwise run this algorithm.[2]

All these approaches, when applied to the median, take at least some multiple of $(n \log n)$ units of time.[3] As you know, the best sorting algorithms take $O(n \log n)$ time also. As you may not know, there is actually a sense in which one can prove every comparison-based sorting algorithm takes $\Omega(n \log n)$ time This raises the natural question of whether it is possible to do selection any faster than sorting. In other words, is the problem of finding the median element of a set significantly easier than the problem of ordering (sorting) the whole set.

Recursive Selection Algorithm

Suppose for a minute that we magically knew how to find the median in $O(n)$ time. That is, we have a routine MagicMedian, that given as input a set A, returns the median. We could then use this in a divide and conquer algorithm for Select as follows;

```
Select($A, i, n$)
(selects the $i$th smallest element in set $A$, where $n = |A|$)
```

[2]We also note that the running time can be improved to $O(n + i \log n)$ by first creating a *heap*, which takes $O(n)$ time, and then performing a Delete-Min operation i times.

[3]An alternate notation for $f(x) = O(g(x))$ is $g(x) = \Omega(f(x))$. Notice the change in roles of f and g. In this notation, we say that all of these algorithms take $\Omega(n \log n)$ time.

```
(1)  if ($n = 1$)
(2)       return the one item in $A$
(3)  else
(4)       $p = MagicMedian(A)$
(5)       Let $H$ be the set of elements greater than $p$
(6)       Let $L$ be the set of elements less than or equal to $p$
(7)       if ($i \le |L|$)
(8)            Return Select($L, i, |L|$)
(9)       else
(10)           Return Select($H, i - |L|, |H|$)
```

By H we do not mean the elements that come after p in the list, but the elements of the list which are larger than p in the underlying ordering of our set. This algorithm is based on the following simple observation. If we could divide the set A up into a "lower half" (L) and an "upper" half (H), then we know which of these two sets the ith smallest element in A will be in. Namely, if $i \le \lceil n/2 \rceil$, it will be in L, and otherwise it will be in H. Thus, we can recursively look in one or the other set. We can easily partition the data into two sets by making two passes, in the first we copy the numbers smaller than p into L, and in the second we copy the numbers larger than p into H.[4]

The only additional detail is that if we look in H, then instead of looking for the ith smallest, we look for the $i - \lceil n/2 \rceil$th smallest, as H is formed by removing the $\lceil n/2 \rceil$ smallest elements from A. We can express the running time by the following recurrence:

$$T(n) \le T(n/2) + cn \tag{5.11}$$

From the master theorem, we know any function which satisfies this recurrence has $T(n) = O(n)$.

So we can conclude that if we already know how to find the median in linear time, we can design a divide and conquer algorithm that will solve the selection problem in linear time. This is nothing to write home about (yet)!

Sometimes a knowledge of solving recurrences can help us design algorithms. First, let's consider what recurrences have only solutions $T(n)$ with $T(n) = O(n)$. In particular, consider recurrences of the form $T(n) \le T(n/b) + cn$, and ask when they have solution $T(n) = O(n)$. Using the master theorem, we see that as long as $\log_b 1 < 1$ (and since $\log_b 1 = 0$ for any b, this means than any b allowed by the master theorem works; that is, any $b > 1$ will work), all solutions to this recurrence will have $T(n) = O(n)$. (Note that b does not have to be an integer.) Interpreting this as an abstract algorithm, and letting $b' = 1/b$, this says that as long as we can solve a problem of size n by first solving (recursively) a problem of size $b'n$, for any $b' < 1$, and then doing $O(n)$ additional work, our algorithm will run in $O(n)$ time. Interpreting this in the selection problem, it says that as long as we can, in $O(n)$ time, choose p to ensure that both L and H have size at most $b'n$, we will have a linear time algorithm. In particular, suppose that, in $O(n)$ time, we can choose p to ensure that both L and H have size at most $(3/4)n$. Then we will be able to solve the selection problem in linear time.

Now suppose instead of a MagicMedian box, we have a much weaker Magic Box, one which only guarantees that it will return some number in the middle half of our set. That is, it will

[4]We can do this more efficiently, and "in place", using the partition algorithm of quicksort.

return a number that is guaranteed to be somewhere between the $n/4$th smallest number and the $3n/4$th smallest number. We will call this box a MagicMiddle box, and can use it in the following algorithm:

```
Select1(A,i,n)
(selects the $i$th smallest element in set $A$, where $n = |A|$ )
(1)  if ($n = 1$)
(2)       return the one item in $A$
(3)  else
(4)       $p = MagicMiddle(A)$
(5)       Let $H$ be the set of elements greater than $p$
(6)       Let $L$ be the set of elements less than or equal to $p$
(7)       if ($i \leq |L|$)
(8)            Return Select1($L, i, |L|$)
(9)       else
(10)           Return Select1($H, i - |L|, |H|$)
```

The algorithm Select1 is similar to Select. The only difference is that p is now only guaranteed to be in the middle half. Now, when we recurse, the decision of the set on which to recurse is based on whether i is less than or equal to $|L|$. The element p is called a *partition element*, because it is used to partition our set A into the two sets L and H.

This is progress, as we now don't need to assume that we can find the median in order to have a linear time algorithm, we only need to assume that we can find one number in the middle half of the set. This seems like a much simpler problem, and in fact it is. Thus our knowledge of which recurrences solve to $O(n)$ led us towards a more plausible algorithm.

Unfortunately, we don't know a straightforward way to even find an item in the middle half. We will now describe a way to find it, however, in which we select a subset of the numbers and then *recursively* find the median of that subset.

More precisely consider the following algorithm (in which we assume that $|A|$ is a multiple of 5.)

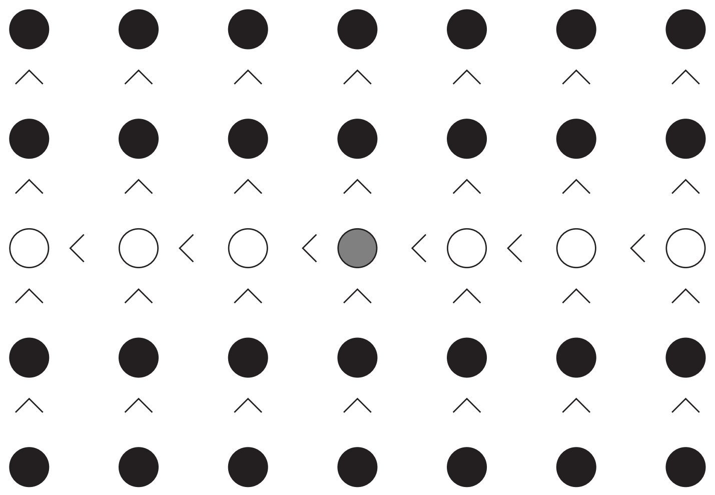
```
MagicMiddle(A)
(1)  Let $n = |A|$
(2)  if ($n < 60$)
(3)       return the median of $A$
(4)  else
(5)       Break $A$ into $k = n/5$ groups of size 5, $G_1, \ldots, G_k$
(6)       for $i = 1$ to $k$
(7)            find $m_i$, the median of $G_i$
(8)       Let $M = \{m_1, \ldots, m_k\}$
(9)       return Select1 ($M, \lceil k/2 \rceil, k$)
```

In this algorithm, we break A into $n/5$ sets of size 5, and then find the median of each set. We then (using Select1 recursively) find the median of medians and return this as our p.

Lemma 5.11 *The value returned by MagicMiddle(A) is in the middle half of A.*

Proof: Consider arranging the elements in the following manner. For each set of 5, list them in sorted order, with the smallest element on top. Then line up all $n/5$ of these lists, ordered by their medians, smallest on the left. We get the picture in Figure 5.9 In this picture, the medians

Figure 5.9: Dividing a set into $n/5$ parts of size 5, finding the median of each part and the median of the medians.



are in white, the median of medians is cross-hatched, and we have put in all the inequalities that we know from the ordering information that we have. Now, consider how many items are less than or equal to the median of medians. Every smaller median is clearly less than the median of medians and, in its 5 element set, the elements smaller than the median are also smaller than the median of medians. Now in Figure 5.10 we circle a set of elements that is guaranteed to be smaller than the median of medians. In one fewer than half the columns, we have circled 3 elements and in one column we have circled 2 elements. Therefore, we have circled at least[5]

$$\left(\frac{1}{2}\left(\frac{n}{5}\right)-1\right)3+2=\frac{3n}{10}-1$$

elements.

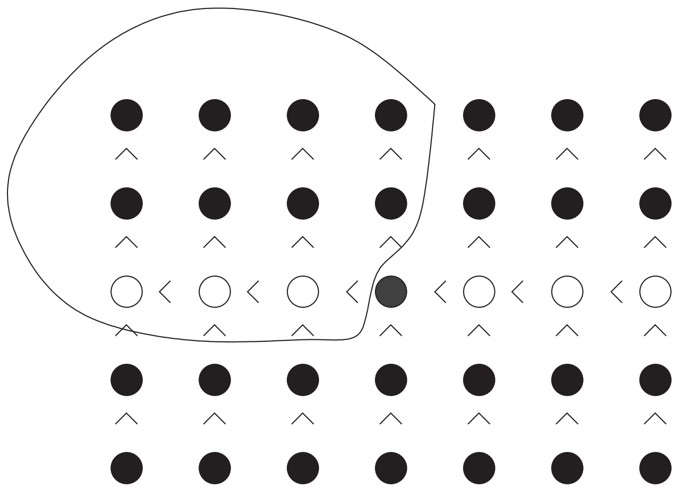
So far we have assumed n is an exact multiple of 5, but we will be using this idea in circumstances when it is not. If it is not an exact multiple of 5, we will have $\lceil n/5\rceil$ columns (in particular more than $n/5$ columns), but in one of them we might have only one element. It is possible that column is one of the ones we counted on for 3 elements, so our estimate could be two elements too large.[6] Thus we have circled at least

$$\frac{3n}{10}-1-2=\frac{3n}{10}-3$$

[5]We say "at least" because our argument applies exactly when n is even, but underestimates the number of circled elements when n is odd.

[6]A bit less than 2 because we have more than $n/5$ columns.

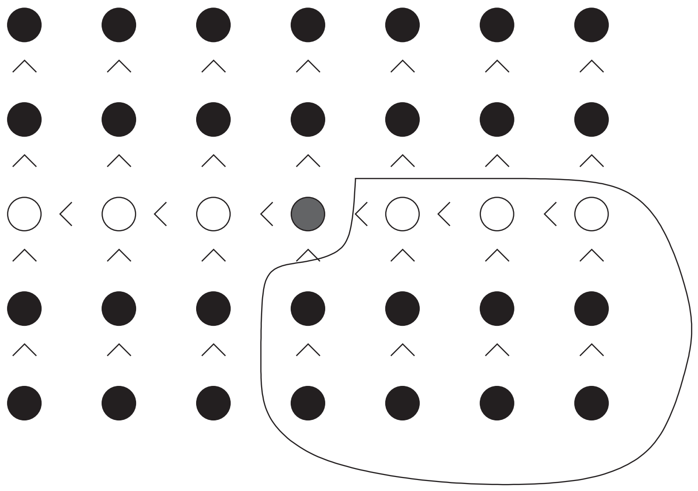
Figure 5.10: The circled elements are less than the median of the medians.



elements. It is a straightforward argument with inequalities that as long as $n \geq 60$, this quantity is at least $n/4$. So if at least $n/4$ items are guaranteed to be less than the median, then at most $3n/4$ items can be greater than the median, and hence $|H| \leq 3n/4$.

We can make the same argument about larger elements. A set of elements that is guaranteed to be larger than the median of medians is circled in the Figure 5.11: By the same argument as

Figure 5.11: The circled elements are greater than the median of the medians.



above, this shows that the size of L is at most $3n/4$. ■

Note that we don't actually identify all the nodes that are guaranteed to be, say, less than the median of medians, we are just guaranteed that the proper number exists.

Since we only have the guarantee that MagicMiddle gives us an element in the middle half of the set if the set has at least sixty elements, we modify Select1 to start out by checking to see if

$n < 60$, and sorting the set to find the element in position i if $n < 60$. Since 60 is a constant, this takes a constant amount of time.

Exercise 5.4-3 Let $T(n)$ be the running time of Select1 on n items. What is the running time of Magic Middle?

Exercise 5.4-4 What is a recurrence for the running time of Select1?

Exercise 5.4-5 Can you prove by induction that each solution to the recurrence for Select1 is $O(n)$?

The first step of MagicMiddle is to divide the items into sets of five; this takes $O(n)$ time. We then have to find the median of each set. There are $n/5$ sets and we spend no more than some constant time per set, so the total time is $O(n)$. Next we recursively call Select1 to find the median of medians; this takes $T(n/5)$ time. Finally, we do the partition of A into those elements less than or equal to the "magic middle" and those that are not. This too takes $O(n)$ time. Thus the total running time is $T(n/5) + O(n)$. This means that for some n_0 there is a constant $c_0 > 0$ such that the running time is no more than $c_0 n$. Now, even if $n_0 > 60$, there are only finitely many cases between 60 and n_0 so there is a constant c such that for $n \geq 60$, the running tme of Magic Middle is no more than $T(n/5) + cn$.

Now Select1 has to call Magic Middle and then recurse on either L or H, each of which has size at most $3n/4$. Adding in a base case that it takes time no more than some constant d to cover sets of size less than 60, we get the following recurrence for the running time of Select1:

$$T(n) \leq \begin{cases} T(3n/4) + T(n/5) + cn & \text{if } n \geq 60 \\ d & \text{if } n < 60. \end{cases} \tag{5.12}$$

We can now verify by induction that $T(n) = O(n)$. What we want to prove is that there is a constant k such that $T(n) \leq kn$. What the recurrence tells us is that there are constants c and d such that $T(n) \leq T(3n/4) + T(n/5) + cn$ if $n \geq 60$, and otherwise $T(n) \leq d$. For the base case we have $T(n) \leq d \leq dn$ for $n < 60$, so we choose k to be at least d and then $T(n) \leq kn$ for $n < 60$. We now assume that $n \geq 60$ and $T(m) \leq km$ for values $m < n$, and get

$$\begin{aligned} T(n) &\leq T(3n/4) + T(n/5) + cn \\ &\leq 3kn/4 + 2kn/5 + cn \\ &= 19/20kn + cn \\ &= kn + (c - k/20)n\ . \end{aligned}$$

As long as $k \geq 20c$, this is at most kn; so we simply choose k this big and by the principle of mathematical induction, we have $T(n) < kn$ for all positive integers n.

Uneven Divisions

This kind of recurrence is actually an instance of a more general class which we will now explore.

Exercise 5.4-6 We already know that when $g(n) = O(n)$, then every solution of $T(n) = T(n/2) + g(n)$ satisfies $T(n) = O(n)$. Use the master theorem to find Big-O bounds to all solutions of $T(n) = T(cn) + g(n)$ for any constant $c < 1$, assuming that $g(n) = O(n)$.

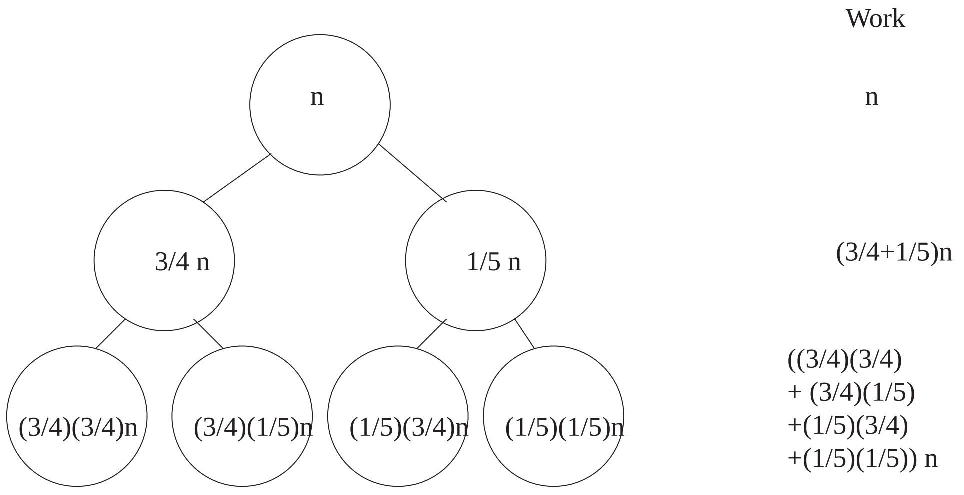
Exercise 5.4-7 Use the master theorem to find Big-O bounds to all solutions of $T(n) = 2T(cn) + g(n)$ for any constant $c < 1/2$, assuming that $g(n) = O(n)$.

Exercise 5.4-8 Suppose $g(n) = O(n)$ and you have a recurrence of the form $T(n) = T(an) + T(bn) + g(n)$ for some constants a and b. What conditions on a and b guarantee that all solutions to this recurrence have $T(n) = O(n)$?

Using the master theorem, for Exercise 5.4-6, we get $T(n) = O(n)$, since $\log_{1/c} 1 < 1$. We also get $O(n)$ for Exercise 5.4-7, since $\log_{1/c} 2 < 1$ for $c < 1/2$. You might now guess that as long as $a + b < 1$, any solution to the recurrence $T(n) \leq T(an) + T(bn) + cn$ has $T(n) = O(n)$. We will now see why this is the case.

First, let's return to the recurrence we had, $T(n) = T(3/4n)+T(n/5)+g(n)$, were $g(n) = O(n)$ and let's try to draw a recursion tree. This doesn't quite fit our model, as the two subproblems are unequal (thus we can't even write down the problem size on the left), but we will try to draw it anyway and see what happens. As we draw levels one and two, we see that at the level one, we

Figure 5.12: Attempting a recursion tree for $T(n) = T(3/4n) + T(n/5) + g(n)$.



have (3/4+1/5)n work. At the level two we have $((3/4)^2+2(3/4)(1/5)+(1/5)^2)n$ work. Were we to work out the third level we would see that we have $((3/4)^3+3(3/4)^2(1/5)+3(3/4)(1/5)^2+(1/5)^3)n$. Thus we can see a pattern emerging. At level one we have $(3/4 + 1/5)n$ work. At level 2 we have, by the binomial theorem, $(3/4+1/5)^2n$ work. At level 3 we have, by the binomial theorem, $(3/4 + 1/5)^3n$ work. And thus at level i of the tree we have $\left(\frac{3}{4} + \frac{1}{5}\right)^i n = \left(\frac{19}{20}\right)^i$ work. Thus summing over all the levels, we get that the total amount of work is

$$\sum_{i=0}^{O(\log n)} \left(\frac{19}{20}\right)^i n \leq \left(\frac{1}{1 - 19/20}\right) n = 20n.$$

We have actually ignored one detail here. In contrast to a recursion tree in which all subproblems at a level have equal size, the "bottom" of the tree is more complicated. Different branches of the tree will reach 1 and terminate at different levels. For example, the branch that follows all 3/4's will bottom out after $\log_{4/3} n$ levels, while the one that follows all 1/5's will bottom out after $\log_5 n$ levels. However, the analysis above *overestimates the work*, that is, it assumes that nothing bottoms out until $\log_{20/19} n$ levels, and in fact, the upper bound we gave on the sum "assumes" that the recurrence never bottoms out.

We see here something general happening. It seems as if to understand a recurrence of the form $T(n) = T(an) + T(bn) + g(n)$, with $g(n) = O(n)$, we can study the simpler recurrence $T(n) = T((a+b)n) + g(n)$ instead. This simplifies things (in particular, it lets us use the Master Theorem) and allows us to analyze a larger class of recurrences. Turning to the median algorithm, it tells us that the important thing that happened there was that the sizes of the two recursive calls, namely $3/4n$ and $n/5$, summed to less than 1. As long as that is the case for an algorithm, the algorithm will work in $O(n)$ time.

Important Concepts, Formulas, and Theorems

1. *Median.* The *median* of a set (with an underlying order) of n elements is the element that would be in position $\lceil n/2 \rceil$ if the set were sorted into a list in order.

2. *Percentile.* The pth percentile of a set (with an underlying order) is the element that would be in position $\frac{p}{100}n$ if the set were sorted into a list in order.

3. *Selection.* Given an n-element set with some underlying order, the problem of *selection* of the ith smallest element is that of finding the element that would be in the ith position if the set were sorted into a list in order. Note that often i is expressed as a fraction of n.

4. *Partition Element.* A *partition element* in an algorithm is an element of a set (with an underlying order) which is used to divide the set into two parts, those that come before or are equal to the element (in the underlying order), and the remaining elements. Notice that the set as given to the algorithm is not necessarily (in fact not usually) given in the underlying order.

5. *Linear Time Algorithms.* If the running time of an algorithm satisfies a recurrence of the form $T(n) \le T(an) + cn$ with $0 \le a < 1$, or a recurrence of the form $T(n) \le T(an) + T(bn) + cn$ with a and b nonnegative and $a + b < 1$, then $T(n) = O(n)$.

6. *Finding a Good Partition Element.* If a set (with an underlying order) has sixty or more elements, then the procedure of breaking the set into pieces of size 5 (plus one leftover piece if necessary), finding the median of each piece and the finding the median of the medians gives an element guaranteed to be in the middle half of the set.

7. *Selection algorithm.* The Selection algorithm with a linear time running guarantee sorts a set of size less than sixty to find the element in the ith position; otherwise it uses the median of medians of five to find a partition element, uses that partition element to divide the set into two pieces and looks for the appropriate element in the appropriate piece recursively.

Problems

1. In the MagicMiddle algorithm, suppose we broke our data up into $n/3$ sets of size 3. What would the running time of Select1 be?

2. In the MagicMiddle algorithm, suppose we broke our data up into $n/7$ sets of size 7. What would the running time of Select1 be?

3. Let
$$T(n) = \begin{cases} T(n/3) + T(n/2) + n & \text{if } n \geq 6 \\ 1 & \text{otherwise,} \end{cases}$$
and let
$$S(n) = \begin{cases} S(5n/6) + n & \text{if } n \geq 6 \\ 1 & \text{otherwise.} \end{cases}$$
Draw recursion trees for T and S. What are the big-O bounds we get on solutions to the recurrences? Use the recursion trees to argue that, for all n, $T(n) \leq S(n)$.

4. Find a (big-O) upper bound (the best you know how to get) on solutions to the recurrence $T(n) = T(n/3) + T(n/6) + T(n/4) + n$.

5. Find a (big-O) upper bound (the best you know how to get) on solutions the recurrence $T(n) = T(n/4) + T(n/2) + n^2$.

6. Note that we have chosen the median of an n-element set to be the element in position $\lceil n/2 \rceil$. We have also chosen to put the median of the medians into the set L of algorithm Select1. Show that this lets us prove that $T(n) \leq T(3n/4) + T(n/5) + cn$ for $n \geq 40$ rather than $n \geq 60$. (You will need to analyze the case where $\lceil n/5 \rceil$ is even and the case where it is odd separately.) Is 40 the least value possible?