APS105 Reading for Lecture 27
Quicksort

Dan Zingaro

March 29, 2010

Partitions and Quicksort

- Quicksort uses a partition procedure to arrange elements of an array so that
  - Elements smaller than the chosen pivot element come first, and
  - Elements at least as large as the pivot follow
- If we could then separately sort these two pieces of the array, the original array would be entirely sorted
- We sort these pieces of the array recursively

Implementing Quicksort

- The base case for the recursion will be the array consisting of zero or one elements
- For the recursive case, the plan is as follows
  - Choose a pivot (we will choose the rightmost value)
  - Partition the array around this pivot value
  - Swap the pivot value so that it sits between the small values and the large values
  - Recursively sort the elements less than the pivot
  - Recursively sort the elements greater than or equal to the pivot

Quicksort: the Code

```
void quicksort (int low, int high, int a[]) {
  if (low < high) {
    int pivot = a[high];
    int i = partition (low, high - 1, pivot, a);
    swap (a, i, high);
    quicksort (low, i - 1, a);
    quicksort (i + 1, high, a);
  }
}
```

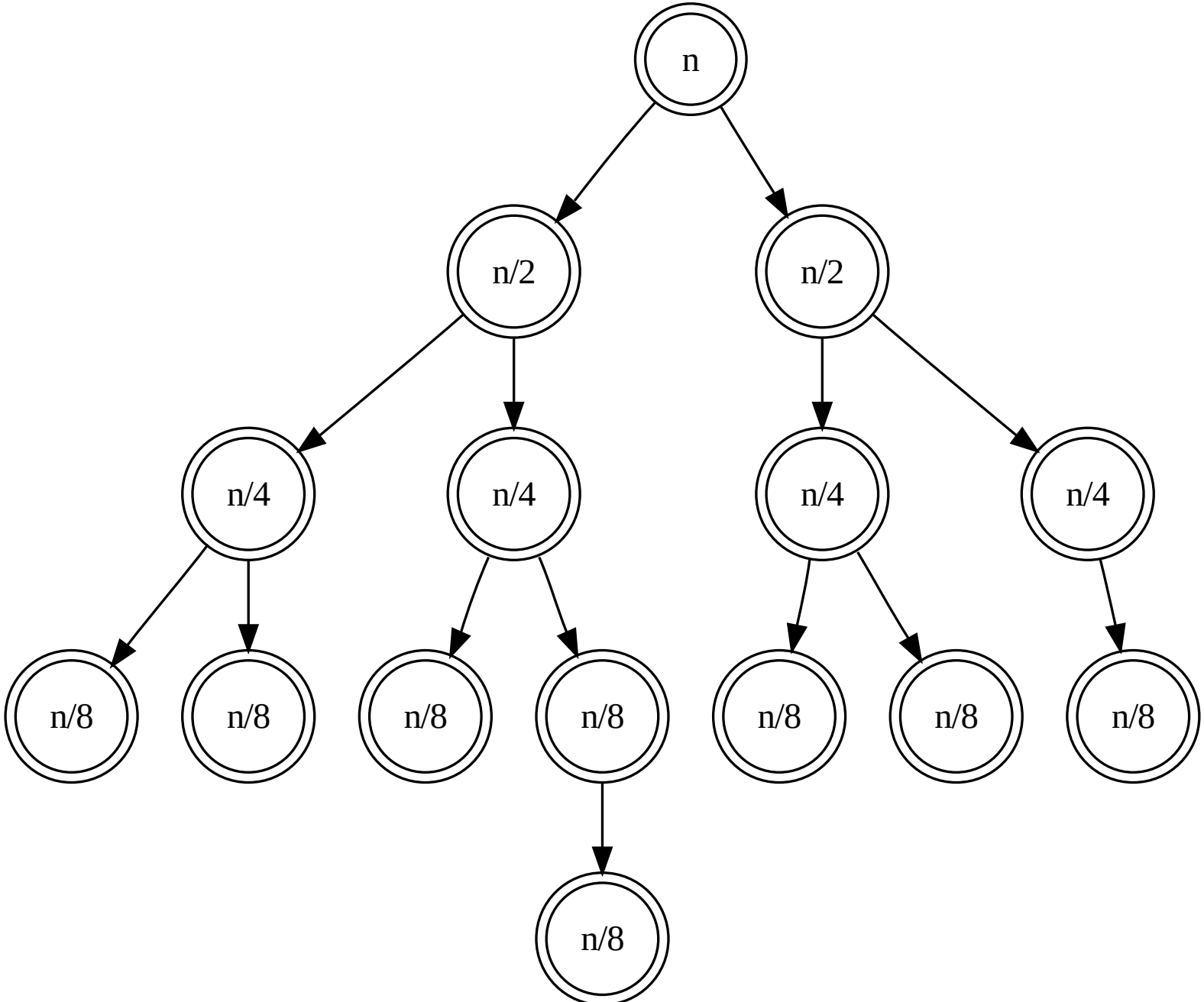
Why Selection Sort is Slow

- One way to look at sorting algorithms is through the number of comparisons they make
- On the first iteration of selection sort, we make `n-1` comparisons
- On the second iteration, we make `n-2` comparisons
- On the final iteration, we make `1` comparison
- In total, we make `1+2+...+n-1` comparisons
- There is a formula for this sequence: $\frac{1}{2}(n^2 - n)$

Best Case for Quicksort

- Assume that we choose a pivot that partitions the list exactly in half on each recursive call
- On the outermost level, the partition step takes n time (one step per element in the list)
- On the two resulting subproblems of size $n/2$, the total partition time is $n/2 + n/2 = n$
- On the resulting four subproblems of size $n/4$ (two from each of the two $n/2$ subproblems), the total partitioning time is still n, and so on
- On each level of recursion, we do a total of n work
- There are $\lg n$ levels of recursion, each of which takes n work
- Total work: $n * \lg n$

Best Case for Quicksort: Tree



Worst Case for Quicksort

- So, in the best case, we take $n * \lg n$ steps
- However, it's always possible that we choose a pivot that partitions the array badly
- Consider: we pass a sorted array to quicksort and choose the rightmost element as the pivot
- On each recursive call operating on an array of size n, we would partition it into an array of $n - 1$ elements and an array of just 1 element
- This worst-case is as bad as selection sort!
- Fortunately, this happens infrequently — usually, quicksort is much faster