Programming Abstractions

CS106B

Cynthia Lee

Stanford University

Today's Topics

Sorting!

1. **The warm-ups**
   - Selection sort
   - Insertion sort
2. **Let's use a data structure!**
   - Heapsort
3. **Divide & Conquer**
   - Merge Sort (aka Professor Sorting the Midterms Using TAs and SLs Sort)
   - Quicksort

Selection Sort

A classic “My First Sorting Algorithm” sorting algorithm

Stanford University

Selection Sort

Compare the **best-case** and **worst-case** costs of this algorithm (tight Big-O characterization of each):

A. Best case = Worst case

B. Best case < Worst case

Why? Explain very specifically.

```
void sort(Vector<int>& vec) {
  int n = vec.size();
  // already-fully-sorted section grows
  // 1 at a time from left to right
  for (int lh = 0; lh < n; lh++) {
    int rh = lh;
    // find the min element in the
    // entire unsorted section
    for (int i = lh + 1; i < n; i++) {
      // found new min?
      if (vec[i] < vec[rh]) rh = i;
    }
    // swap min into sorted section
    int tmp = vec[lh];
    vec[lh] = vec[rh];
    vec[rh] = tmp;
  }
}
```

Bubble Sort

It’s not very good, famously so…
https://www.youtube.com/watch?v=k4RRi_ntQc8
(arguably better than Selection Sort though!)

Insertion Sort

Another classic “Beginner” sorting algorithm

Stanford University

Insertion Sort

Compare the **best-case** and **worst-case** costs of this algorithm (tight Big-O characterization of each):

A. Best case = Worst case

B. Best case < Worst case

Why? Explain very specifically.

```
void sort(Vector<int>& vec) {
  int n = vec.size();
  // already-sorted section grows 1 at a
  // time from left to right
  for (int i = 1; i < n; i++) {
    int j = i;
    // does this item needs to move
    // left to be in order?
    while (j > 0 && vec[j-1] > vec[j]) {
      // keep swapping this item with
      // its left neighbor if it is
      // smaller than the left neighbor
      int tmp = vec[i];
      vec[i] = vec[j];
      vec[j] = tmp;
      j--;
    }
  }
}
```

Heap Sort

Stanford University

Heapsort

Pretty simple!!

1. Take the unsorted array and insert each element into a heap priority queue
2. While the queue is not empty, dequeue an element from the heap priority queue

The elements come out of the priority queue in sorted order.

Fun fact: you don’t need extra array storage, you can do this *in place* in the original array.

Professor’s Sorting Algorithm

Sorting in the “real world”

Preliminary Step: We need a “combine two sorted piles” algorithm

Start: you have two piles, each of which is sorted

- Take the overall smallest element (smallest in either pile) and add that one element to the combined-sorted pile
- Repeat until the two starting piles are empty and the combined-sorted pile is complete
- *Towards the end, you might end up with one pile already empty and the other not, so just move from non-empty pile into combined-sorted pile*

Preliminary Step:
We need a “combine two sorted piles” algorithm

Start: you have two piles, each of which is sorted

- **Take the overall smallest element (smallest in either pile) and add that one element to the combined-sorted pile**
- Repeat until the two starting piles are empty and the combined-sorted pile is complete
- *Towards the end, you might end up with one pile already empty and the other not, so just move from non-empty pile into combined-sorted pile*

How many elements do we examine to find the overall smallest element?

How many steps does it take to **merge** two sorted sub-piles, A and B?

In other words, how long does it take to do the “combine two sorted piles” algorithm on piles A and B? (best/tight answer)

A. O(log(|A|+|B|)) steps
B. O(|A|+|B|) steps
C. O(|A+B|)^2 steps
D. O(|A|^2 + |B|^2)steps
E. Other/none/more than one

Professor’s sorting algorithm:

Stanford CS classes can have more than 500 students! Sorting the midterms alphabetically to prepare for handing them back is a non-trivial task. Luckily, I don’t have to do it myself…

1. **Find two grad students**, give each half of the unsorted midterms
2. Tell the grad students to sort their own pile, then give it back
3. Combine the two piles into one sorted pile, using our simple combine algorithm
4. Done!

Grad student’s sorting algorithm:

Sorting ~250 exams is still a non-trivial task! Luckily, the grad students don’t have to do it themselves!

1. **Find two SLs**, give each half of the unsorted midterms
2. Tell the SLs to sort their own pile, then give it back to you
3. Combine the two piles into one sorted pile, using our simple combine algorithm
4. Done! *(give your sorted pile to professor)*

SL’s sorting algorithm:

1. **Find two students**, give each half of the unsorted midterms
2. Tell the students to sort their own pile, then give it back to you
3. Combine the two piles into one sorted pile, using our simple combine algorithm
4. Done! *(give sorted pile to grad student)*

Student’s sorting algorithm:

1. **Find two visiting prospective freshmen,** give each half of the unsorted midterms
2. Tell the profros to sort their own pile, then give it back to you
3. Combine the two piles into one sorted pile, using our simple combine algorithm
4. Done! *(give sorted pile to SL)*

Prospective Frosh’s sorting algorithm:

1. **By now, the pile only has zero or one exam in it** *(for the sake of this example, assume the starting number of exams makes this true at this point)*
2. Done! *(give sorted pile to student)*

Consider an arbitrarily chosen (generic) particular exam and mentally track its progress throughout the algorithm.

How many times does your exam pass through the **merge** algorithm?

A. 1 time
B. 2 times
C. O(logn) times
D. O(n) times
E. Other/none/more than one

BigO Analysis of Mergesort

Every paper is merged log(n) times

- This is the number of times we can divide the stack of n papers by 2 before we can't divide anymore

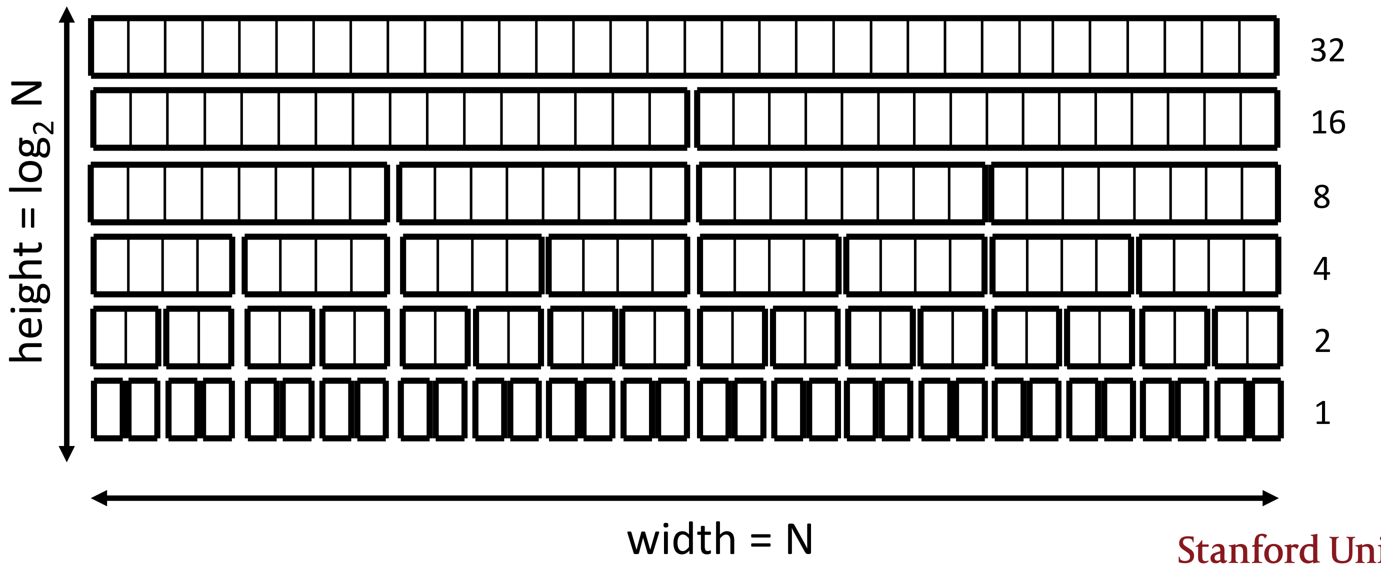
There are n papers

O(nlogn)

Merge Sort runtime intuition

Merge sort performs O(N) operations on each level. *(width)*

- Each level splits the data in 2, so there are $\log_2 N$ levels. *(height)*
- Product of these = $N * \log_2 N$ = O($N \log N$). *(area)*
- Example: N = 32. Performs ~ $\log_2 32$ = 5 levels of N operations each:



Merge Sort

- Compare the best case and worst case of Merge sort:

  A. Best case = Worst case

  B. Best case < Worst case

  Why? Explain very specifically in terms of the structure of the code.

Quicksort

Divide & Conquer

Imagine we want students to line up in alphabetical order to pick up their midterms, which (as we know from Professor sorting algorithm!) are sorted in alphabetical order.

1. “Everybody in the first half of the alphabet, go over there!” “Everybody in the second half, go over there!”
   - At this point, we at least have some kind of division based on ordering, but it’s very crude. Each of the two “over there” groups is completely unsorted within the group, but...
2. …at least now you have two groups that are each smaller and easier to sort, <u>*so recursively sort each half*</u>.

That’s it!*

* ok, actually there are some details…

Divide & Conquer

Imagine we want students to line up in alphabetical order to pick up their midterms, which (as we know from Professor sorting algorithm!) are sorted in alphabetical order.

1. "Everybody in the **first half** of the alphabet, go over there!" "Everybody in the second half, go over there!"
   - At this point, we at least have some kind of division based on ordering, but it's very crude. Each of the two "over there" groups is completely unsorted within the group, but...
2. …at least now you have two groups that are each smaller and easier to sort, *so recursively s*

That's it!*

* ok, actually there are some details

Rather than doing the work of finding the *actual* median, we just choose an arbitrary or random element to be the divider. Say, the first array index of the group, or randomly select an array index from the group.

Quicksort

- Consider the best case and worst case of Quicksort (best/tight characterization in each case)

  A. Best case = Worst case

  B. Best case < Worst case

  Why? Explain very specifically in terms of the structure of the code.