Sorting 3: Merge Sort

Instructors: Sam McCauley and Dan Barowy

March 11, 2022

Admin

- Practice midterm, topics, etc. posted
- Remember to bring questions next Wednesday
- Any questions?

Side Note on Logarithms And Strong Induction

Logarithms

- Definition: if $b^i = a$, then $i = \log_b a$.
- Identities: $\log_b a = \frac{log_2 a}{\log_2 b}$, $\log_b a^i = i \log_b a$, $\log_b(x \cdot y) = \log_b x + \log_b y$
- Can we show that $\log_{10} x = O(\log_2 x)$?
- Can we show that $\log_2 x = O(\log_{10} x)$?
- In general: we'll just write $O(\log n)$ because any constant base doesn't change the asymptotics

Logarithms: Some Intuition

- $\log_2 n$ is the number of times n needs to be divided by 2 to reach a value ≤ 2. (Why?)
- $\log_2 n$ is very close to the number of digits of n in binary
  - Similarly, $\log_{10} n$ is close to the number of digits of n in base 10.
- For all $n \geq 1$, we have $\log_2 n \leq n$. (How could we prove this?)

Strong Induction

- So far: in an induction, we assume the inductive hypothesis for some k, and use it to prove the inductive hypothesis for $k + 1$.
- But if we want, we can actually assume *more* than that
- We can assume: the inductive hypothesis holds for all i from 1 to k (after all, we used the ladder to get here)
- What can we do with this?

Strong Induction Example

- Show that for all $n \geq 15$, change of n cents can be made using 3 cent and 7 cent coins
- Base case?
    - $n = 15$ seems like a good choice
    - Let's actually come back to this…
- Inductive hypothesis: for some $k \geq 15$, change of k cents can be made using 3 and 7 cent coins.
- Idea of induction?
    - We can make $k + 1$ cents if we can make $(k + 1) - 3$ cents
    - How can we ensure that we have already addressed this value?

Using Strong Induction

- Inductive hypothesis (*Strong* Induction): for some $k \geq 17$, for all i from 15 to k, change of i cents can be made using 3 and 7 cent coins.
- Assume the inductive hypothesis (by strong induction). The inductive hypothesis implies that it is possible to make change for $(k+1)-3$ cents. Adding a 3 cent piece gives us $k+1$ cents.
- Base case? Need to show how to make $15, 16, 17$ cents.
    - 15: use 5 3-cent coins
    - 16: use 3 3-cent coins and 1 7-cent coin
    - 17: use 1 3-cent coin and 2 7-cent coins

Returning to Binary Search

Binary Search

- Search for an item in a sorted array
- Idea: compare to middle. If we’re searching for a smaller element recurse on left half of array; otherwise recurse on right half of array
- Base case: on an array of size 1, return true if element is equal to the query element; otherwise return false
- Can we prove correctness?
- Can we prove running time?

Example Binary Search Code

```java
public int binarySearch(int[] arr, int query){
   return binarySearchHelper(arr, query, 0, arr.length);
}
//Invariant: if query in arr then arr[low] <= query < arr[high]
public int binarySearchHelper(int[] arr, int query, int low, int high){
   if(low == high - 1) {
      if(query == arr[low]) return low;
      else return -1;
   }
   int mid = (low + high)/2;
   if(query < arr[mid])
      return binarySearchHelper(arr, query, low, mid);
   else
      return binarySearchHelper(arr, query, mid, high);
}
```

Binary search: correctness

- By induction!
- Base case?
    - Binary search works correctly when `low == high - 1`
- Inductive hypothesis?
    - For some k, binary search with `low - high` $= k$ will correctly find any item between `arr[low]` and `arr[high]`

Binary search: correctness

- Inductive Step:
- Let's say we binary search in an array of size $k+1$
- Since we assume the array is sorted, if the value we are searching for is smaller than the middle element, it must be located in the left half of the array.
- (Same argument if larger)
- Assume by *strong induction* that the inductive hypothesis holds for all $1 \leq i \leq k$. That means it holds for $(k+1)/2$
- Therefore, our recursive call correctly returns if the element is in the correct half of the array. That means binary search is correct

Induction Discussion

- Could we have used weak induction (i.e. not strong induction) here?
    - No—we needed the algorithm to be correct on an array of size $(k+1)/2$, not an array of size k
    - (It is possible to use weak induction with a change of variable.)
- Strong vs weak induction:
    - Both prove the inductive hypothesis for $k+1$
    - Weak induction: only assume I.H. for k to prove $k+1$
    - Strong induction: assume I.H. for $1, 2, \ldots k$ to prove $k+1$ (may use some base case other than 1
    - Never *need* to use weak induction
- Weak induction is good when you grow by 1 every time
- Recursive algorithms often use strong induction if they reduce the input size by more than 1.

Binary Search: Running Time

- How much time does each recursive call take?
- $O(1)$: just need to compare and recurse
- How many times does the algorithm recurse?
- Array size decreases by 2 each recursive call. Therefore, can recurse at most $O(\log n)$ times
- $O(\log n)$ running time

Try/Catch Statements

Try catch statements

- Can *catch* an exception in Java before it stops the program
- The program doesn't need to stop running!
- Can give more detailed diagnostic, or can fix the error so the program can continue
- Can also be helpful for debugging
- You use the keyword `try` to show the code you want to run that may generate an error. Then, you write `catch`: the code you want to run if the error is generated
- Let's see an example

Merge Sort

Merge Sort

- We can sort faster than $O(n^2)$!
- Merge sort is a classic way to do that
- Very fast in practice; used as the basis of many state of the art sorting algorithms

Sorting Recursively?

- Can we use recursion to sort?
- Base case?
    - One-element array is always sorted.
- How can we create a smaller subproblem?
    - Could do one element smaller, but that gives us $O(n^2)$ (like in selection sort)
    - Can we divide the size by 2 like in Binary Search?

Merge Sort Hint

- Let’s say I have two sorted arrays
- How fast can I sort the concatenation of these arrays?

-3	17	21	40

-4	10	11	13

Goal:

-4	-3	10	11	13	17	21	40

- Hint: where does the *smallest* element between the two arrays reside?
- Repeatedly take the smaller of the first remaining element in the two arrays. Takes $O(n)$ time

Merge Sort

- If array has size 1, just return
- Split array into two halves
- Sort each half
    - How?
    - Using Merge Sort!
- Then merge the resulting arrays together in $O(n)$ time by walking through them
- Let’s do an example on the board

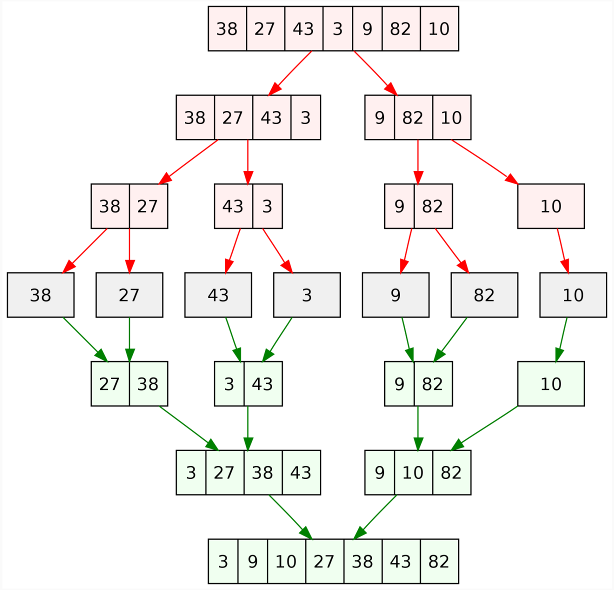
Merge Sort Code

```
void mergeSortRecursive(int data[], int temp[], int low, int high) {
   int n = high-low+1;
   int middle = low + n/2;
   int i;
   if (n < 2) return;
   // move lower half of data into temporary storage
   for (i = low; i < middle; i++) {
      temp[i] = data[i];
   }
   mergeSortRecursive(temp,data,low,middle-1);
   mergeSortRecursive(data,temp,middle,high);
   // merge halves together
   merge(data,temp,low,middle,high);
}
```

Merge temp[low..middle-1] and data[middle..high]

```
void merge(int data[], int temp[], int low, int middle, int high) {
   int ri = low; // result index
   int ti = low; // temp index
   int di = middle; // destination index
   // while two lists are not empty merge smaller value
   while (ti < middle && di <= high) {
      if (data[di] < temp[ti]) {
         data[ri++] = data[di++]; // smaller is in high data
      } else {
         data[ri++] = temp[ti++]; // smaller is in temp
      }
   }
   // possibly some values left in temp array
   while (ti < middle) {
      data[ri++] = temp[ti++];
   }
}
```

Merge Sort Image


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82