Lecture 9: More Sorting

CS 62

Fall 2018

Alexandra Papoutsaki & William Devanny

Assignment 3

- What to do when you want to sort data that cannot fit in memory of your computer?
  - On-disk sorting

- Break data into chunks that will fit in memory, sort chunks, copy into new files: `0.tempfile`, `1.tempfile` , …
- Keep `ArrayList` of files
- Merge files together until one big sorted file.
- Note: You can't keep file open as both read and write!

Assignment 3 and Lab 3

- Read info on File I/O in Java and file systems in appendix to assignment.
- See on-line Streams cheat sheet

- Lab 3: More complexity/timing (sorting)

Merge Sort

- Example of Divide & Conquer algorithm
  - Divide array in half
  - Sort each half
  - Merge halves together into completely sorted array

- *Needs extra space (not in-place)*
- Stable: two objects with equal keys appear in the same order in **sorted** output as they appear in the input unsorted array.

MergeSort

```
/**
* MergeSort   Sorts data >= low and < high
* @param list data to be sorted
* @param low start of the data to be sorted
* @param high end of the data to be sorted (exclusive)
*/
private void mergeSort(int[] data, int low, int high){
    if( high-low > 1 ){
        int mid = low + (high-low)/2;
        mergeSort(data, low, mid);
        mergeSort(data, mid, high);
        merge(data, low, mid, high);
      }
}
```

```
/** Merge data >= low and < high into sorted data.
* Data >= low and < mid are in sorted order.
* Data >= mid and < high are also in sorted order
*/
public void merge(int[] data, int low, int mid, int high){
int[] temp = new int[high-low]; // make temporary array temp of size high-low
int k = 0, i = low, j = mid;
while( i < mid && j < high ){
    if( data[i] <= data[j]){
        temp[k] = data[i];
        i++;
    }else{
        temp[k] = data[j];
        j++;
    }
    k++;
}
// copy over the remaining data on the low to mid side if there is some remaining.
// copy over the remaining data on the mid to high side if there is some remaining.
// Only one of these two while loops should actually execute
// copy the data back from temp to array
```

```
// copy over the remaining data on the low to mid side if there is some remaining.
while(i < mid){
        temp[k] = data[i];
        k++;
        i++;
}
// copy over the remaining data on the mid to high side if there is some remaining.
while(j < high){
        temp[k] = data[j];
        k++;
        j++;
}
// Only one of these two while loops should actually execute

// copy the data back from temp to array
for(int index = 0; index < temp.length; index++ ){
        data[index+low]=temp[index];
}
```

Example

Sort: 85 24 63 45 17 31 96 50 (whiteboard)

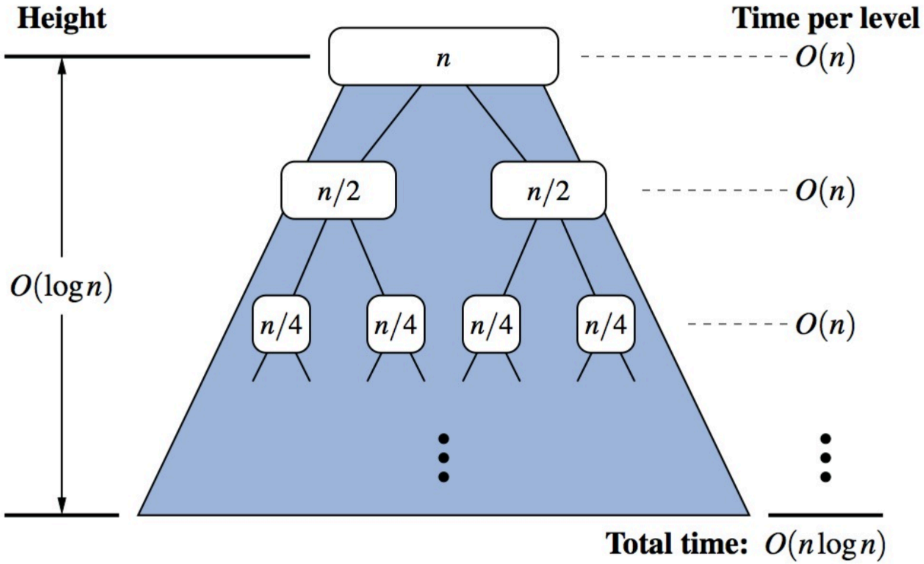
Correctness

- $P(n)$: If $high - low = n$ then $mergeSort(data, low, high)$ will result in $data[low\,..\,high]$ being correctly sorted
- For simplicity, assume $merge$ is correct
- Assume $P(k)\ for\ all\ k < n$, show $P(n)$
- If $n = 0$ or 1 then (correctly) do nothing
- Assume $n > 1$
  - Call $mergeSort(data, low, mid)$ and $mergeSort(data, mid + 1, high)$ where $mid = low + (high - low)/2.$
  - Hence $mid - low < n, high - (mid + 1) < n$
  - By induction $data[low..mid]$ and $data[mid + 1\,..\,high]$ now sorted.
  - call $merge(data, low, mid, high)$ and, by assumption on $merge$, $data[low\,..\,high]$ now sorted! Thus $P(n)$ true.

Complexity

- Claim: $mergeSort$ is $O(n \log n)$
  - where log is base 2
- Merge of two lists of combined size n takes $\leq n - 1$ comparisons.
  - Think of merging [1,3,5,7] and [2,4,6,8]
- If l levels:
  - $n/2^l = 1$
  - $n = 2^l$
  - $l = \log n$

- $\log n$ levels
- each taking $O(n)$ operations
- $O(n \ \log n)$ in total



Complexity

- $P(m)$: if data has 2^m elements then $mergesort$ makes $< m \ * \ 2^m$ total comparisons.
- Assume $P(k)\ for\ all\ k\ <\ 2^m$. Prove $P(m)$
- $P(0), P(1)$ clear. Show $P(m)$
- Sort first half, second half, and then merge
- Each half has size $^{2^m}/_2 = 2^{m-1} < 2^m$, so by induction, each takes $< (m-1) * 2^{m-1}$ comparisons
- Therefore total number of comparisons in $mergesort$
  $$< (m-1) * 2^{m-1} + (m-1) * 2^{m-1} + (2^m - 1)$$
  $$= (m-1) * 2^m + (2^m - 1) = m * 2^m - 1 < m * 2^m$$
- Thus $P(m)$ is true
- If $n = 2^m$ then $mergeSort$ takes $n \log n$ comparisons ($m \ = \ \log n$).