Merge Sort

CS206
April 16

Timing

Table 1

size	selection	Insertion	Insertion	Heap
1000	16	15	11	2
2000	8	12	26	3
4000	24	23	20	5
8000	96	95	81	10
16000	370	378	315	17
32000	1585	1359	1218	36
64000	5771	5590	4605	77
128000	23087	21547	19849	161
256000				345
512000				1128
1024000				1973
2048000				3225
4096000				7577
8192000				18586

10000==1 second

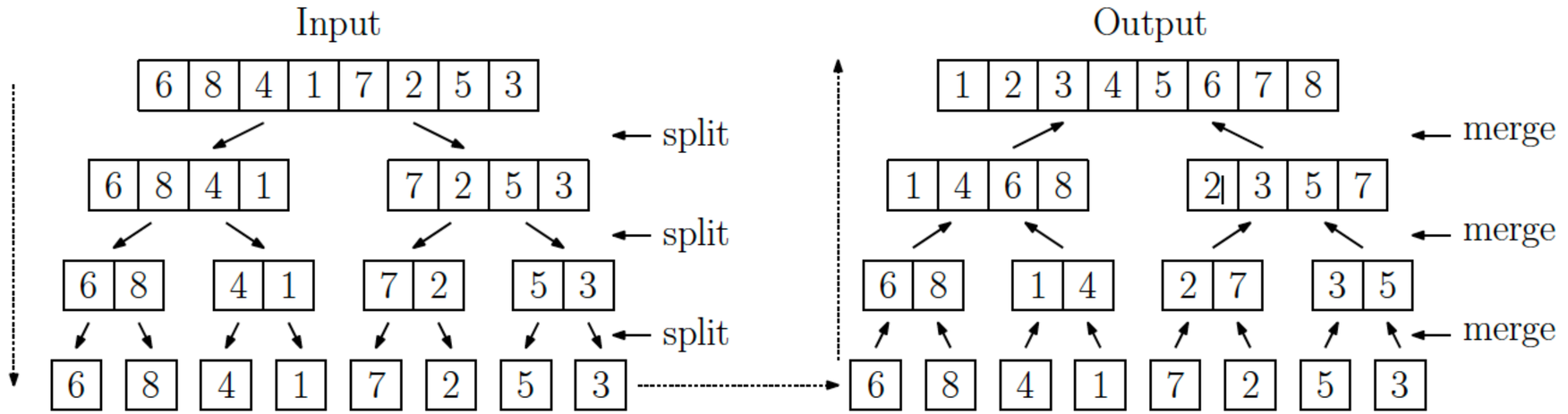
Divide-and-Conquer

- Divide – the problem (input) into smaller pieces
- Conquer – solve each piece individually, usually recursively
- Combine – the piecewise solutions into a global solution (if needed)
- Usually involves recursion
- For example .. binary search

Merge Sort

- Sort a sequence of numbers A, $|A| = n$
- Base: $|A| = 1$, then it's already sorted
- General
  - divide: split A into two halves, each of size $\frac{n}{2}$ ($\left\lfloor \frac{n}{2} \right\rfloor$ and $\left\lceil \frac{n}{2} \right\rceil$)
  - conquer: sort each half (by calling mergeSort recursively)
  - combine: merge the two sorted halves into a single sorted list

Example


Input
Output
6 8 4 1 7 2 5 3
split
split
split
merge
merge
merge
1 2 3 4 5 6 7 8
1 4 6 8
2 3 5 7
6 8
1 4
2 7
3 5
6 8 4 1 7 2 5 3

Algorithm

```
mergeSort(S):
   if S.size() <= 1 return
   else
      s1 = S[0,n/2]
      s2 = S[n/2+1,n-1]
      mergeSort(s1)
      mergeSort(s2)
      S = merge(s1, s2)
```

Merge Algorithm

- The key is the merging process
- How does one merge two sorted lists?
- Each element in $A \bigcup B$ is considered once
- $O(n)$

```
Algorithm merge(A, B)
  Input sorted A and B
  Output sorted A ∪ B
  S = empty sequence
  while(!A.isEmpty() and
        !B.isEmpty())
   if A.first() < B.first()
      S.addLast(A.removeFirst())
   els
      S.addLast(B.removeFirst())
 while (!A.isEmpty())
 S.addLast(A.removeFirst())
 while (!B.isEmpty())
 S.addLast(B.removeFirst())
 return S
```

Merge (in Java)

```
private int[] domerge(int[] list1, int[] list2) {
    int[] rtn = new int[list1.length + list2.length];
    int locr=0, loc1=0, loc2=0;
    while (loc1<list1.length && loc2<list2.length)  {
        if (list1[loc1] < list2[loc2])
        {
        rtn[locr++]=list2[loc2++];
        }
        else
        rtn[locr++]=list1[loc1++];
    }
    for (int i=loc1; i<list1.length; i++)
        rtn[locr++]=list1[i];
    for (int i=loc2; i<list2.length; i++)
        rtn[locr++]=list2[i];
    return rtn;
    }
```

MergeSort

```
    public int[] mergesort(int[] list) {
        return doMergeSort(list, 0, list.length-1);
    }

    private int[] doMergeSort(int[] list, int strt, int eend)
    {
    if (eend==strt)
    {
        int[] tmp = new int[1];
        tmp[0]=list[strt];
        return tmp;
    }
    if (eend<strt)
        return new int[0];
    int mid = (strt+eend)/2;
    return domerge(mergesort(list, strt, mid), mergesort(list, mid+1, eend));
    }
```

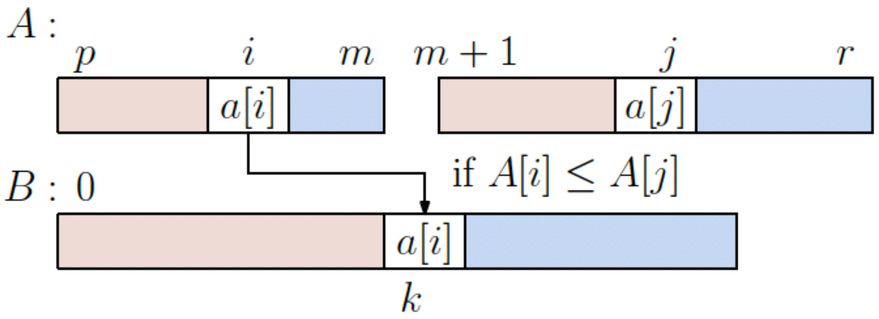
Timing

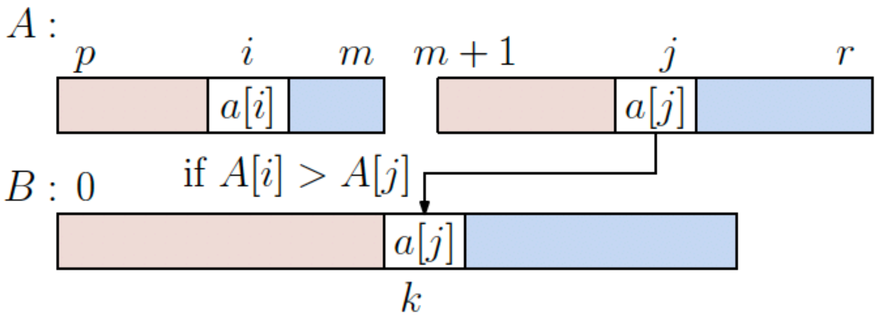
Table 1

size	selection	Insertion	Insertion	Heap	merge
1000	16	15	11	2	3
2000	8	12	26	3	3
4000	24	23	20	5	7
8000	96	95	81	10	13
16000	370	378	315	17	27
32000	1585	1359	1218	36	58
64000	5771	5590	4605	77	119
128000	23087	21547	19849	161	219
256000				345	372
512000				1128	776
1024000				1973	1631
2048000				3225	3822
4096000				7577	6772
8192000				18586	14159

In-place Merge

- Making new lists is slow!
- How does one merge two sorted lists `A[p,…,m]` and `A[m+1,…,r]`?
- Use a temp array `B` and maintain two indices `i` and `j`, one for each subarray





MergeSort using one temp array

```
    private int[] array;
    private int[] tempMergArr;
    private int length;
    public int[] mergesort3(int inputArr[]) {
        this.array = inputArr;
        this.length = inputArr.length;
        this.tempMergArr = new int[length];
        doMergeSort3(0, length - 1);
        return array;
    }

    private void doMergeSort3(int lowerIndex, int higherIndex) {

        if (lowerIndex < higherIndex) {
            int middle = lowerIndex + (higherIndex - lowerIndex) / 2;
            // Below step sorts the left side of the array
            doMergeSort3(lowerIndex, middle);
            // Below step sorts the right side of the array
            doMergeSort3(middle + 1, higherIndex);
            // Now merge both sides
            mergeParts3(lowerIndex, middle, higherIndex);
        }
    }
```

Merge with temp array

```
private void mergeParts3(int lowerIndex, int middle, int higherIndex) {

    for (int i = lowerIndex; i <= higherIndex; i++) {
        tempMergArr[i] = array[i];
    }
    int i = lowerIndex;
    int j = middle + 1;
    int k = lowerIndex;
    while (i <= middle && j <= higherIndex) {
        if (tempMergArr[i] <= tempMergArr[j]) {
            array[k] = tempMergArr[i];
            i++;
        } else {
            array[k] = tempMergArr[j];
            j++;
        }
        k++;
    }
    while (i <= middle) {
        array[k] = tempMergArr[i];
        k++;
        i++;
    }
}
```

Timing

Table 1

size	selection	Insertion	Insertion	Heap	merge	merge (improved)
1000	16	15	11	2	3	2
2000	8	12	26	3	3	3
4000	24	23	20	5	7	7
8000	96	95	81	10	13	9
16000	370	378	315	17	27	16
32000	1585	1359	1218	36	58	32
64000	5771	5590	4605	77	119	69
128000	23087	21547	19849	161	219	143
256000				345	372	294
512000				1128	776	563
1024000				1973	1631	1191
2048000				3225	3822	2412
4096000				7577	6772	5191
8192000				18586	14159	10282

Summary of Sorting Algorithms

Algorithm	Time	Notes
selection-sort		▪ slow ▪ in-place ▪ for small data sets (< 1K)
insertion-sort		▪ slow ▪ in-place ▪ for small data sets (< 1K)
heap-sort		▪ fast ▪ in-place ▪ for large data sets (1K — 1M)
merge-sort		▪ fast ▪ sequential data access ▪ for huge data sets (> 1M)

Mini Homework

14, 6, 18, 2, 13, 7, 8, 9, 3, 17, 5, 10, 11, 12, 15, 19, 16, 0, 1, 4

For the data above,show all the steps of a merge sort, along the lines of slide 5.