[TAP:DMPLA] Selection Sort

```
                                (Vector<Patient> data, Comparator<Patient> c)
public static void selectionSort(~~int[] data~~){
   for (int curN = data.length - 1; curN > 0; curN--) {
     int maxIdx = 0;
     for (int i = 1; i <= curN; i++){
        if ~~(data[i] > data[maxIdx])~~
           maxIdx = i;  ((c.compare(data.get(i)
                                   data.get(maxIdx)) > 0)
     }
     swap(data, maxIdx, curN);
   }
}
public static void main(String[] args){
  Vector<Patient> patients;
  …
  selectionSort(patients, new NameComparator());
}
```

- What would you change if you want to sort a Vector using a Comparator (see the main method)?

A. I’m fully sure
B. I’m partially sure
C. I’m not sure

Administrative Details

- Lab 5 Today
  - Submit partners!
  - Challenging, but shorter and a partner lab – more time for exam prep!
- Mid-term exam is Wednesday, March 14
  - During your normal lab session
  - You’ll have approximately 1 hour & 45 minutes (if you come on time!)
  - Closed-book: Covers Chapters 1-7 & 9, handouts, and all topics up through Sorting
  - A “sample” mid-term **and** study sheet will be available online

Today’s Outline

- Sort
  - Merge Sort
  - Quick Sort

Sorting a Deck of Cards

Time Complexity:

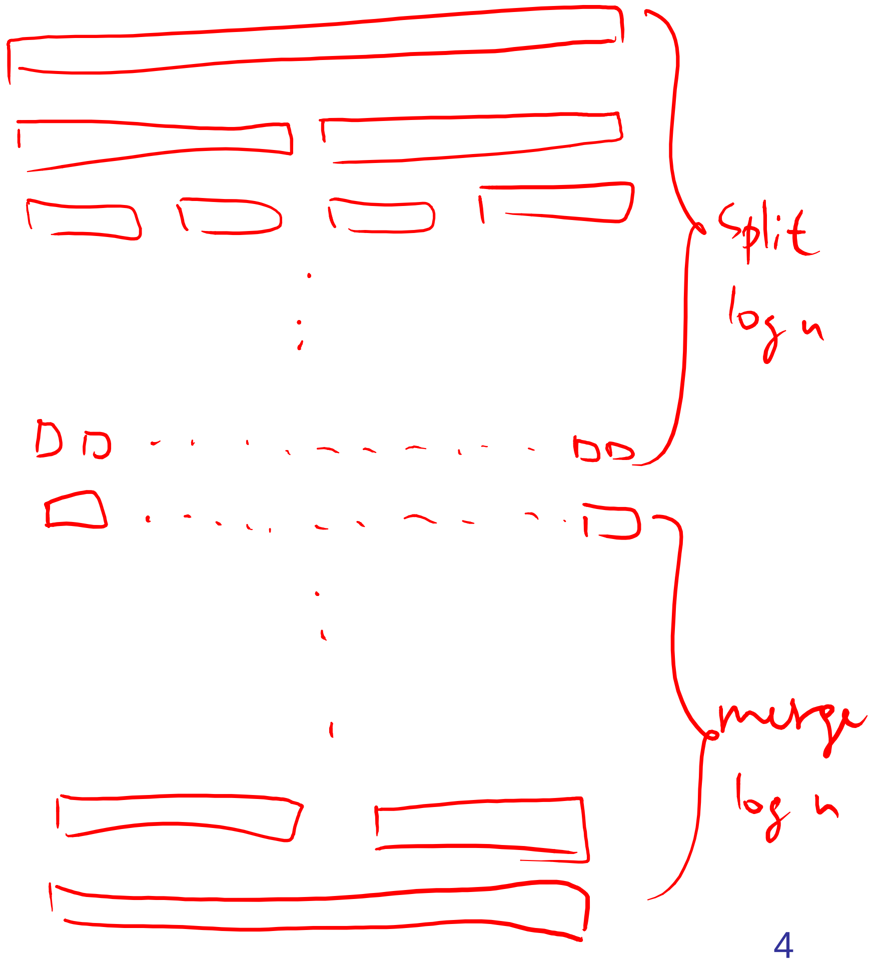
A. O(n)

B. O(n log n)

C. O(n^2)

D. O(n^3)

E. Not sure



Merge Sort

- [8 14 29 1 17 39 16 9]
- [8 14 29 1] [17 39 16 9]
- [8 14] [29 1] [17 39] [16 9]
- [8] [14] [29] [1] [17] [39] [16] [9]
- [8 14] [1 29] [17 39] [9 16]
- [1 8 14 29] [9 16 17 39]
- [1 8 9 14 16 17 29 39]

Merge Sort

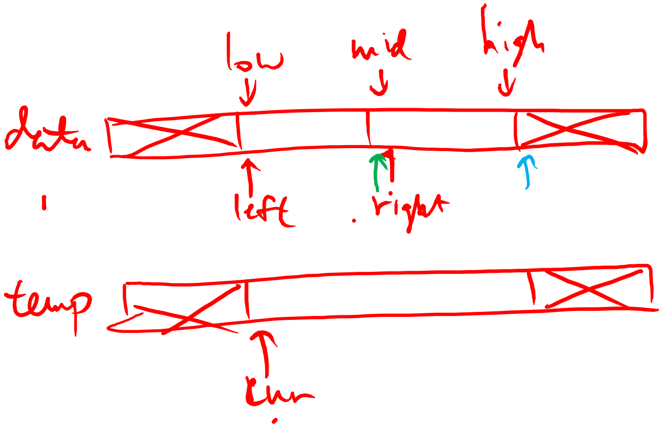
```
public static void mergeSort(int[ ] data) {
    int[] temp = new int[data.length];
    mergeSortHelper( data, 0, data.length-1, temp);
}
public static void mergeSortHelper(int[ ] data, int low, int high, int[] temp) {
    // base case
    if(low >= high)
        return;

    //recursive case
    int mid = (low + high)/2;
    mergeSortHelper(data, low, mid, temp);
    mergeSortHelper(data, mid+1, high, temp);
    merge(data, low, mid+1, high, temp);
}
```

Aside: merge() method

public static void merge(int[ ] data, int low, int mid, int high), int[] temp

~~int[] temp = new int[data.length];~~

```
int left=low;
int right=mid;
int cur=low;

while (left<=mid-1 && right <= high){
    if(data[left]<data[right])
        temp[cur++] = data[left++];
    else
        temp[cur++] = data[right++];
}
while(left <=mid-1)
    temp[cur++] = data[left++];
while (right<=high)
    temp[cur++] = data[right++];

for(int i=low; i<=high; i++)
    data[i]=temp[i];
```



Merge Sort Summary

- Overview (A “divide and conquer” approach)
  - Recursively divide the list in half, until each sublist contains only 1 element (i.e. “sorted”)
  - Recursively (sort-)merge sorted sublists
- Time complexity:
  - Best case: O(n log n)
  - Average case: O(n log n)
  - Worst case: O(n log n)
- Space complexity:
  - O(2n) = O(n)

Today’s Outline

- Sort
  - Merge Sort
  - Quick Sort

Sorting a Deck of Cards

Time Complexity:

A. O(n)

B. O(n log n) ← best , ave

C. $O(n^2)$ ← worst

D. $O(n^3)$

E. Not sure

Quick Sort (random pivot)

- [8 14 29 1 17 39 16 9]
- [8 14 1 9 16 29 17 39]
- [1 8 14 9 16 17 29 39]
- [1 8 9 14 16 17 29 39]

≈ log n

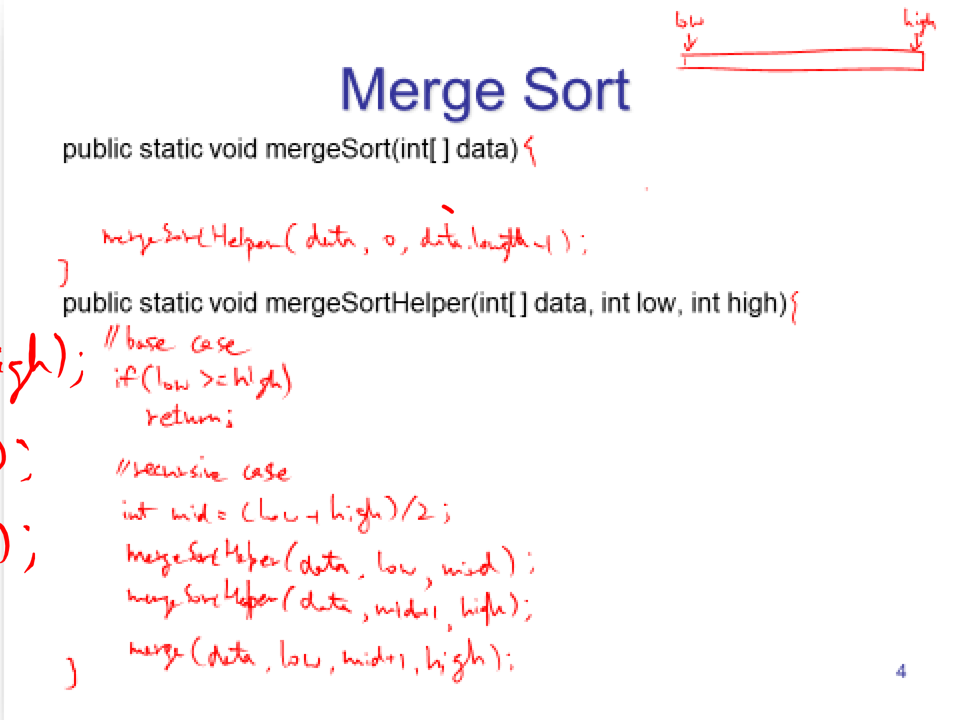
(n in the worse case)

Quick Sort (pivot = left-most element)

- [8 14 29 1 17 39 16 9]
- [1 8 14 29 17 39 16 9]
- [1 8 9 14 29 17 39 16]
- [1 8 9 14 16 17 29 39]
- [1 8 9 14 16 17 29 39]

Quick Sort

```
public static void quickSort(int[ ] data){

    quickSortHelper(data, 0, data.length-1);

}
public static void quickSortHelper(int[] data, int low, int high){
    //base case
    if (low >=high)
        return;
    //recursive case
    int pivotI= partition(data, low, high);
    quickSortHelper(data, low, pivotI-1);
    quickSortHelper(data, pivotI+1, high);

}
```

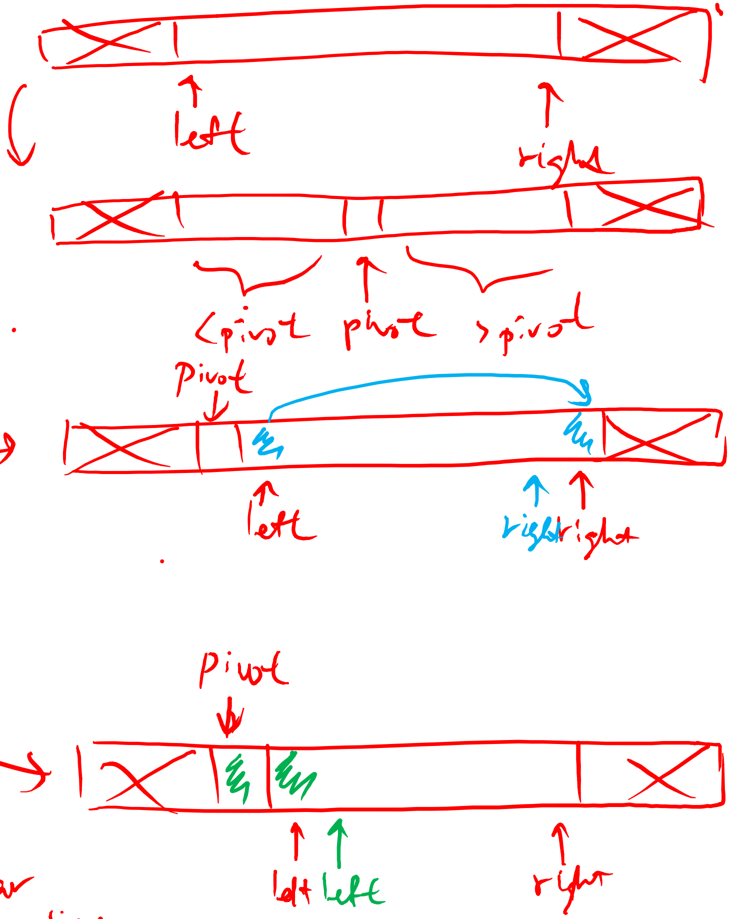


Merge Sort

```
public static void mergeSort(int[ ] data){

    mergeSortHelper(data, 0, data.length-1);
}
public static void mergeSortHelper(int[] data, int low, int high){
    //base case
    if(low >=high)
        return;

    //recursive case
    int mid= (low+high)/2;
    mergeSortHelper(data, low, mid);
    mergeSortHelper(data, mid+1, high);
    merge(data, low, mid+1, high);
}
```

Aside: Partition() method

```
public static int partition(int data[], int left, int right){
    int pivot = data[left];
    left++;

    while (left <= right){
        if (data[left] > pivot){
            swap(data, left, right);
            right--;
        } else {
            data[left-1] = data[left];
            left++;
        }
    }
    data[right]= pivot;
    return right;
}
```



similar to insertion sort

1 2 3 0 4 5

0 1 2 3 4 5

Quick Sort Summary

- Overview
  - Randomly pick a pivot, then move smaller elements to the left and bigger to the right.
  - Recursively sort left and right sublists
- Time complexity:
  - Best case: O(n log n)
  - Average case: O(n log n)
  - Worst case: O(n^2)
- Space complexity:
  - O(n)

Sorting : Time Complexity

Algorithm	Best	Ave	Worst
Bubble	O(n)	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	O(n)	$O(n^2)$	$O(n^2)$
Merge	O(n log n)	O(n log n)	O(n log n)
Quick	O(n log n)	O(n log n)	$O(n^2)$