SOPHISTICATED SORTING ALGORITHMS

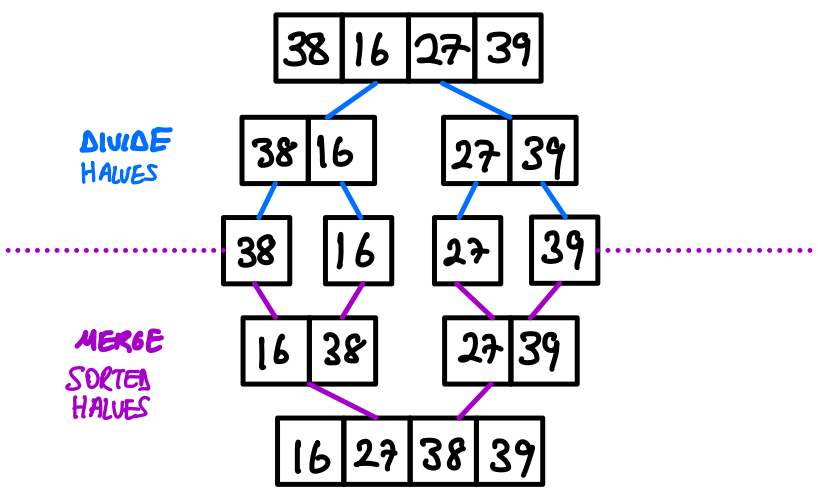
- MERGE SORT
- QUICK SORT
- RADIX SORT

MERGE SORT

- DIVIDE AND CONQUER
- RECURSIVE
  - DIVIDE DATA INTO TWO HALVES
  - SORT EACH HALF
  - MERGE EACH SORTED HALVES

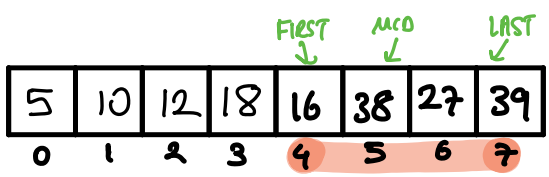
BEST/WORST CASE
O(N logN)

EXAMPLE

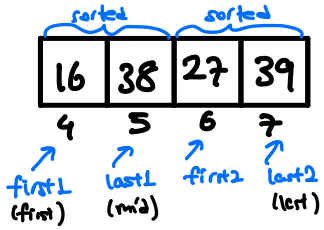


MERGE

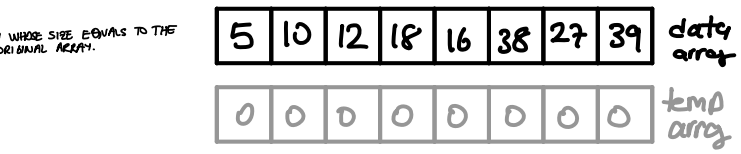
Assume int[] data

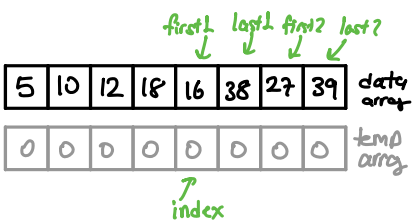


merge(data, 4, 5, 7)



MEMORY USAGE EXTRA MEMORY WHOSE SIZE EQUALS TO THE SIZE OF THE ORIGINAL ARRAY.

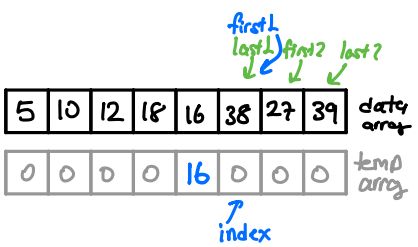




first1<=last1 first2<=last2 ✓

data[first1] < data[first2] ✓

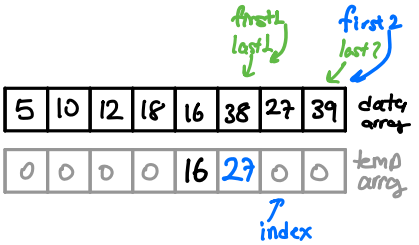
tempLinde<J= data[first1]
first1++;
Indexte;



first1<=last1 first2<=last2 ✓

data[first1] < data[first2] ✗

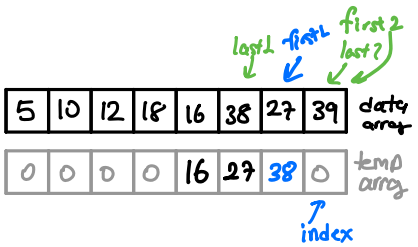
tempLinde<J= data[first2]
Indexte;
first2++;



first1<=last1 first2<=last2 ✓

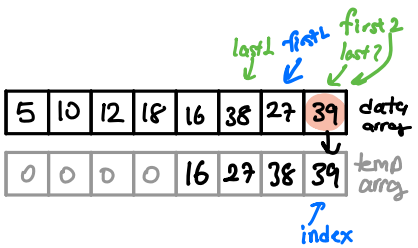
data[first1] < data[first2] ✓

tempLinde<J= data[first1]
first1++;
Indexte;

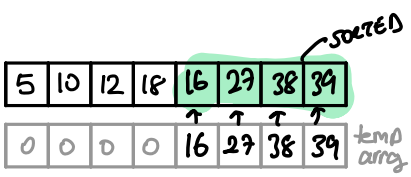


first1<=last1 first2<=last2 ✗

Copy left overs to temp



COPY FROM TEMP TO DATA ARRAY



PSEUDOCODE for MERGE TWO SORTED SUBARRAYS INTO SORTED WHOLE!

```
merge (data, first, mid, last)
        int[] temp = new int [data.length];
        int first1 = first;
        int last1 = mid;

        int first2 = mid+1;
        int last2 = last;

        int index = first1;


        for(; first1<=last1 && first2<=last2; index++)
              if( less( data[first1], data[first2] )){
                    temp[index] = data[first1];
                    first1++;
               }else{
                    temp[index] = data[first2];
                    first2++;
               }
        }

        for(; first1<=last1; index++){
              temp[index] = data[first1];
              first1++;
        }

        for(; first2=last2; index++){
              temp[index] = data[first2];
              first2++;
        }
        for(int i=first; i<=last; i++)
              data[i] = temp[i];
    }
}
```

COPY REMAINING ITEMS IN THE FIRST HALF (IF ANY)

COPY REMAINING ITEMS IN THE SECOND HALF (IF ANY)

COPY FROM TEMP TO DATA

```
mergeSort( data, first, last){
        if(first1 = last){
              int mid = (first+last)/2;
              mergeSort(data, first, mid);
              mergeSort(data, mid+1, last);
              merge(data, first, mid, last);
        }
}
```

QUICK SORT

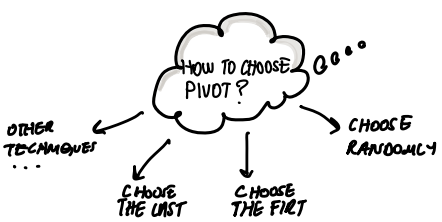
- DIVIDE AND CONQUER
  - PARTITION AN ARRAY IN TWO PARTS
  - SORTS THE PARTS INDEPENDENTLY
  - COMBINES THE SORTED SUBSEQUENCES BY A SIMPLE CONCATENATION

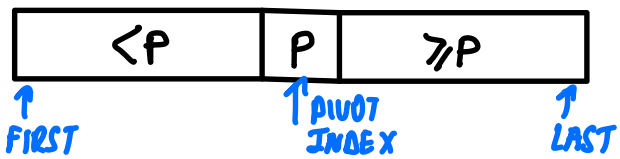
WORST-CASE $O(N^2)$

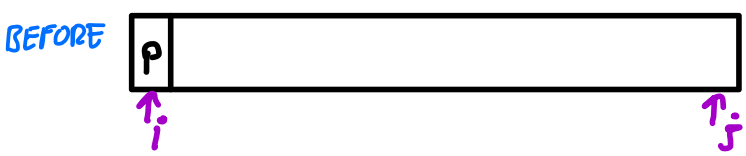
BEST-CASE $O(N \log N)$

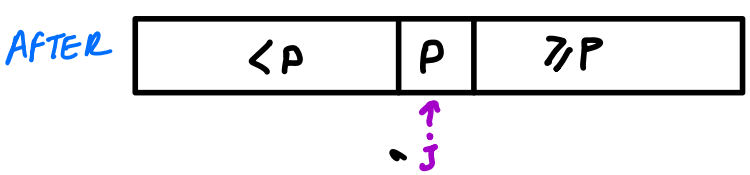
- ALL THE HARD WORK IS DONE BEFORE THE RECURSIVE CALLS

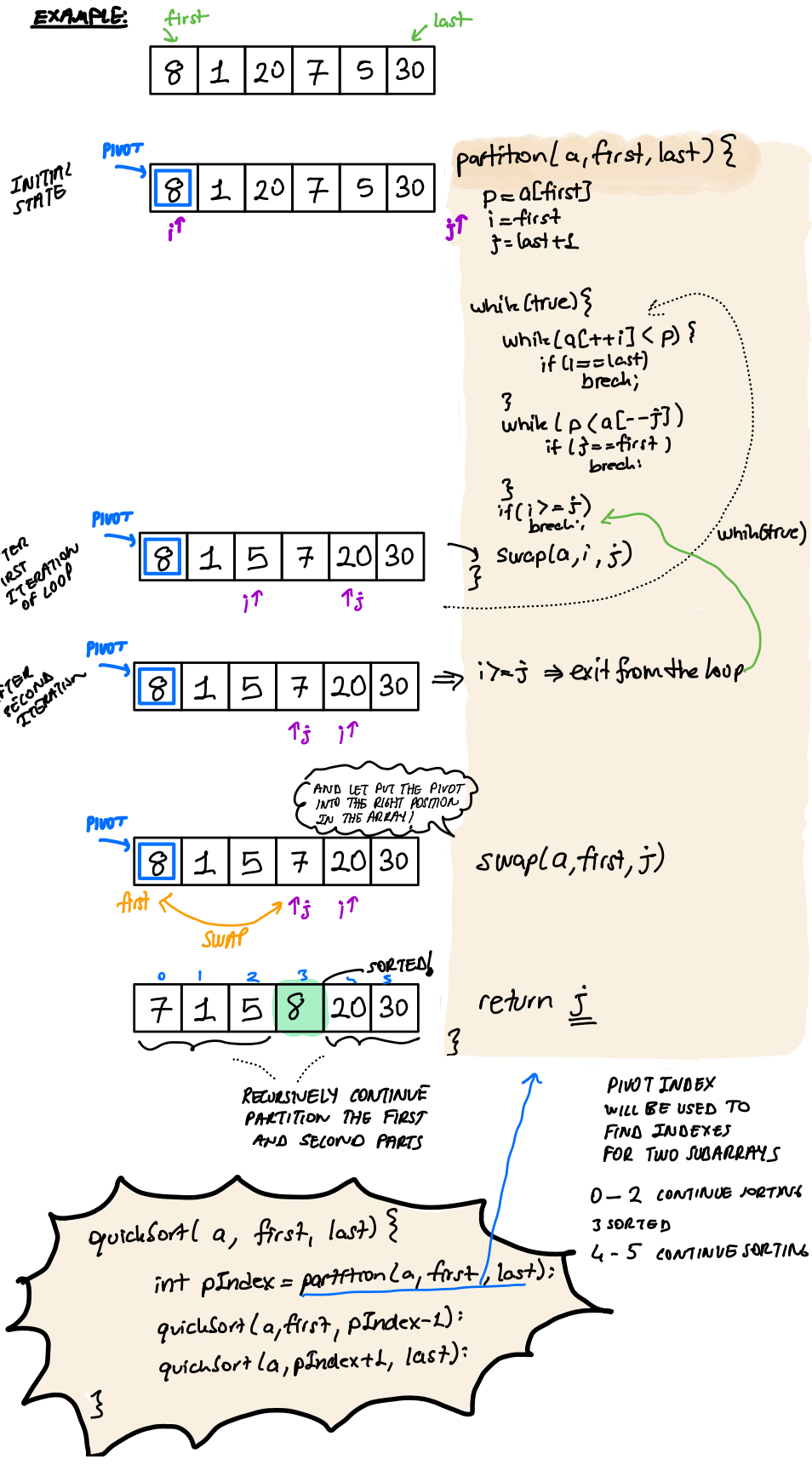
PARTITION PLACES THE PIVOT IN ITS CORRECT PLACE POSITION WITHIN THE ARRAY










EXAMPLE:
first
last
8 1 20 7 5 30
INITIAL STATE
PIVOT
i
j
partition(a, first, last) {
p = a[first]
i = first
j = last + 1
while (true) {
while (a[++i] < p) {
if (i == last)
break;
}
while (p < a[--j])
if (j == first)
break;
}
if (i >= j)
break;
swap(a, i, j)
}
while(true)
AFTER FIRST ITERATION OF LOOP
8 1 5 7 20 30
AFTER SECOND ITERATION
8 1 5 7 20 30
i >= j ⇒ exit from the loop
AND LET PUT THE PIVOT INTO THE RIGHT POSITION IN THE ARRAY!
swap(a, first, j)
first
SWAP
0 1 2 3 4 5
SORTED!
7 1 5 8 20 30
return j
RECURSIVELY CONTINUE PARTITION THE FIRST AND SECOND PARTS
PIVOT INDEX WILL BE USED TO FIND INDEXES FOR TWO SUBARRAYS
0 – 2 CONTINUE SORTING
3 SORTED
4 - 5 CONTINUE SORTING
quickSort(a, first, last) {
int pIndex = partition(a, first, last);
quickSort(a, first, pIndex-1);
quickSort(a, pIndex+1, last);
}

RADIX SORT

BEST-CASE
WORST-CASE
O(N)

- NON-COMPARATIVE SORTING ALGORITHM
- IT AVOIDS COMPARISON BY CREATING AND DISTRIBUTING ELEMENTS INTO BUCKETS ACCORDING TO THEIR RADIX.
- FOR ELEMENTS WITH MORE THAN ONE SIGNIFICANT DIGIT, THIS BUCKET PROCESS IS REPEATED FOR EACH DIGIT WHILE PRESERVING THE ORDER OF THE PRIOR STEP

0123 2154 0222 0004 0283 1560 1061 2150

(1560 2150) (1061) (0222) (0123 0283) (2154 0004) GROUP BY 4TH DIGIT

1560 2150 1061 0222 0123 0283 2154 0004

(0004) (0222 0123) (2150 2154) (1560 1061)(0283) GROUP BY 3RD DIGIT

0004 0222 0123 2150 2154 1560 1061 0283

(0004 1061) (0123 2150 2154) (0222 0283)(1560) GROUP BY 2ND DIGIT

0004 1061 0123 2150 2154 0222 0283 1560

GROUP BY 1ST DIGIT

0004 0123 0222 0283 1061 1560 2150 2154

SORTED