PROGRAM DESCRIPTION

Write several sorts and compare their behavior.

TASKS / REQUIREMENTS

1. **NOTE:** Naming is critical for these tasks and requirements below. If the names are not written those described below exactly, your project will not be graded.

2. Right-click on the **JavaCoreTemplate** project and select Copy. Right-click in an empty area of the Package Explorer and select Paste.

   The project name must follow this pattern:

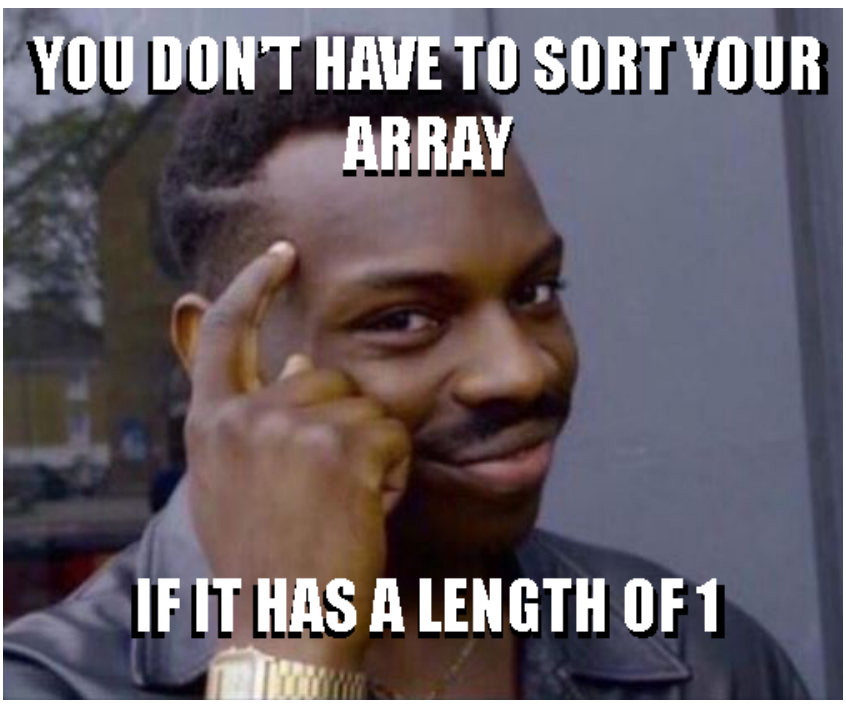
   `{FLnameSortComparison_{TERM}`, where {FLname} is the first letter of your first name plus your last name, and {TERM} is the current semester and year. E.g. if your name is Maria Marciano and its Fall 2025, your project name must be **`MMarciano_SortComparison_F25`**.

3. Change the package name in your project from **`change_this_pkg_name`** to **`sortcomparison`**.

4. BasicSorter tasks:
   1. Download the **Sorter.java** interface and put it in the **`sortcomparison`** package.
   2. Read the comments above each method to understand what they are supposed to do.
   3. Create a class named **`BasicSorter`** in the **`sortcomparison`** package. In the class, Add (interface button, and search for Sorter. Choose the "Inherited abstract methods". Eclipse will create the class and automatically add method stubs for the interface.
   4. You do not modify the **`Sorter`** interface. You add your code to the **`BasicSorter`** class.
   5. **You must write your own sorting code. If you use Arrays.sort(), Arrays.parallelSort(), Collections.sort(), or any other pre-built method to sort the data, the grade for the entire assignment will be 0.** It is OK to use the guidebook or wikipedia or other sites to look for pseudo-code versions of the algorithms, but write the code yourself. It is the only way you will really learn how to implement sort algorithms.

5. Add the unit testing classes. These classes will be used to test the code that you write.
   1. Create a new package in the src folder called **`sbccunittest`**. To be clear: **`sbccunittest`** should be a child of the src folder, not of the **`sortcomparison`** package.
   2. Download **BasicSorterTester.java** into **`sbccunittest`**.

6. Unit Testing
   1. Debug all java compilation Errors (see the Problems view). The unit tests can't be run until these errors are fixed.
   2. In the Package Explorer, right-click on the **`sbccunittest`** package | Run As... | JUnit Test.
   3. Initially, all of the tests will probably fail. However, as you add functionality to the BasicSorter class, tests will begin to pass.
   4. Work on **`BasicSorter.insertionSort`** first.



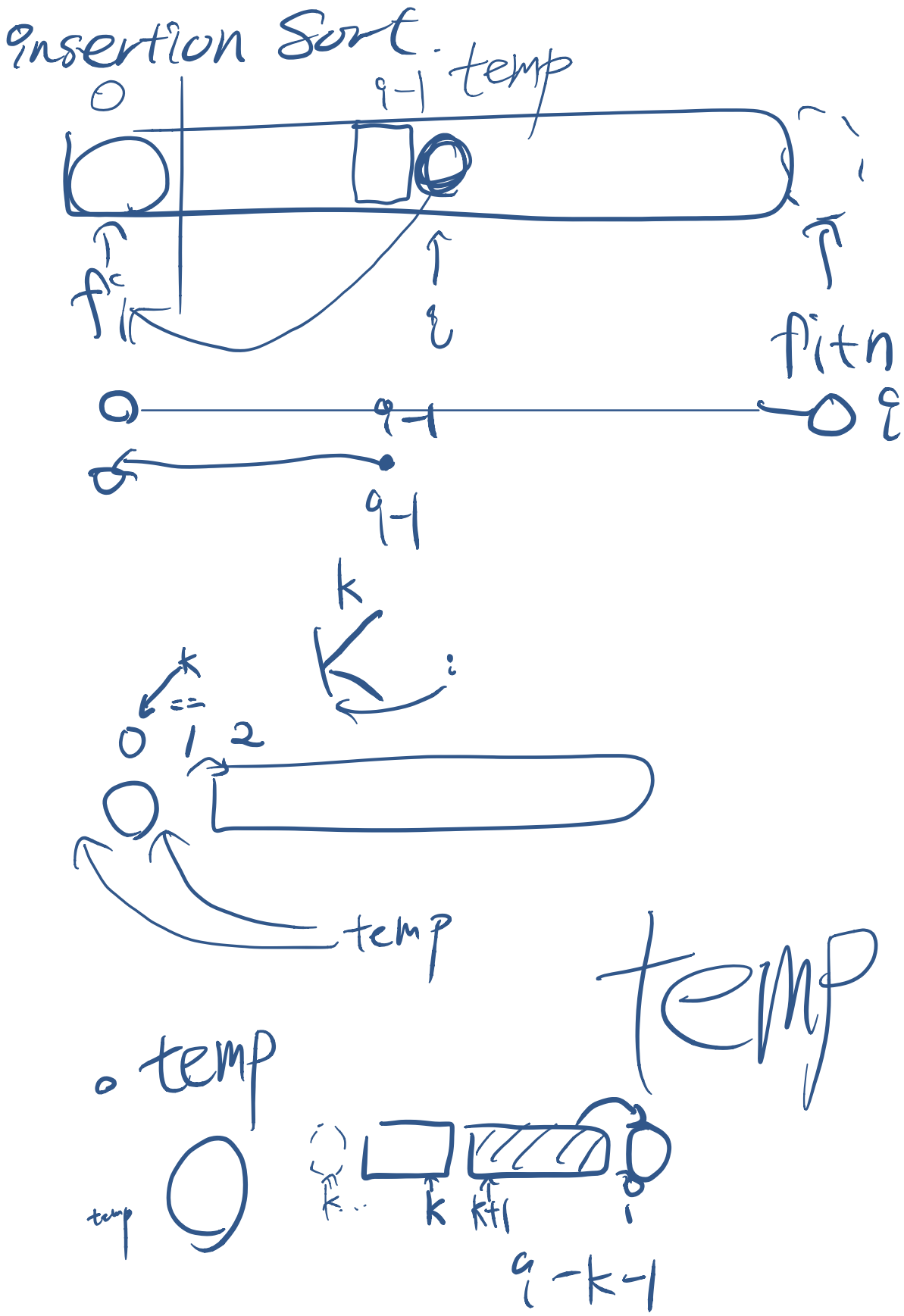
7. There is no user interface requirement for this assignment.

EXTRA CREDIT

Implement heapSort.

SCORING

```
 5 pts  - testInsertionSort
10 pts  - testPartition
10 pts  - testQuickSort
 5 pts  - testMerge
10 pts  - testMergeSort:

1 pt EC - testHeapify
1 pt EC - testHeapSort
```

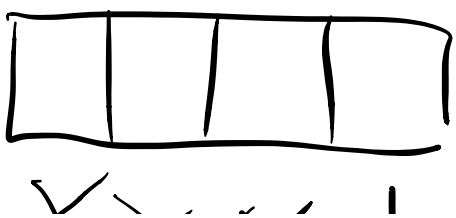

Insertion Sort.
temp
0
q-1
first
i
first+n
q-1
q-1
k
K
i
k
==
0
1
2
temp
temp
temp
temp
k
k+1
i
q-k-1

Quick Sort

Implementation

```
public static void quicksort (int [] data, int first, int n)
// Precondition: data has at least n components
                 starting at data [first]
// Postcondition: The elements of data have been
                  rearranged so that data[first]
                   <= data [first+] <= ... <=
                                        data[first+
                                          n-1]
{
   int pivotIndex; // Array Index for the pivot
                      element.
   int n1; // # elements before the pivot element
   int n2; // # elements after the pivot element

  if (n > 1)
   {
       // Partition the array and set the
          pivot index.
      pivotIndex = partition(data, first, n);
       // Compute the sizes of the two pieces
      n1 = pivotIndex - first;
```

```
        n2 = n - n1 - 1;
        // Recursive calls will now sort the
        two pieces
        quickSort (data, first, n1);
        quickSort (data, pivotIndex+1, n2);
    }
}
```



^ X X 1

D7 D6 D5 D4

P0.4 P5.1 P5.0 P0.5

0

ex P0.5 on

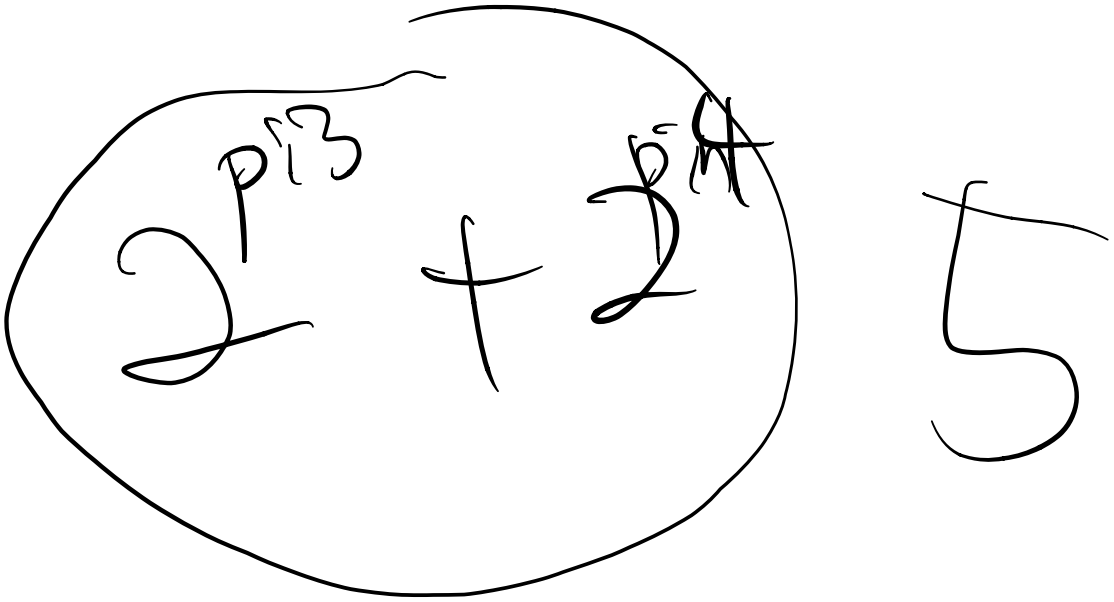
$2^5 = 0x20$

into GPIO_BASE

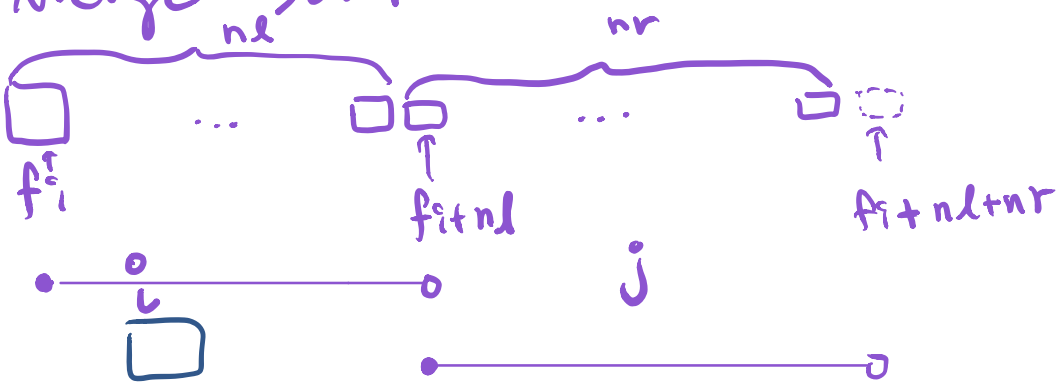
SET0

LDR r0 = GPIO_BASE

STR reg, [r0, #SET0]


$2^{p-3} + 2^{p-1}$
5

Merge Sort




```
i = fi;
j = fi+nl;
k = fi;

while( i < fi+nl && j < fi+nl+nr ) {
    if(array[i] < array[j]) {
        temp[k] = array[i];
        i++;
        k++;

    }
    else if(array[i] == array[j]) {
        temp[k] = array[i];
        i++;
        k++;
        temp[k] = array[j];
        j++;
        k++;
```

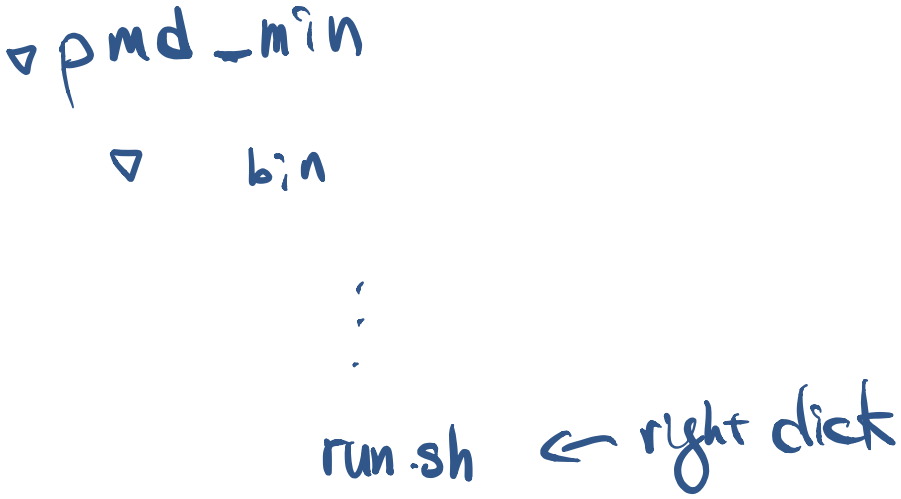
```
else if( array[i] >  array[j] ) {
        temp[k ] = array[j];
        k++;
        j++;
    }

}
while(  i < fi + nl ) {
        temp[k++] = array[i++];
}
while( j < fi+nl+nr) {
        temp[k++] = array[j++];
}
```

With a threshold

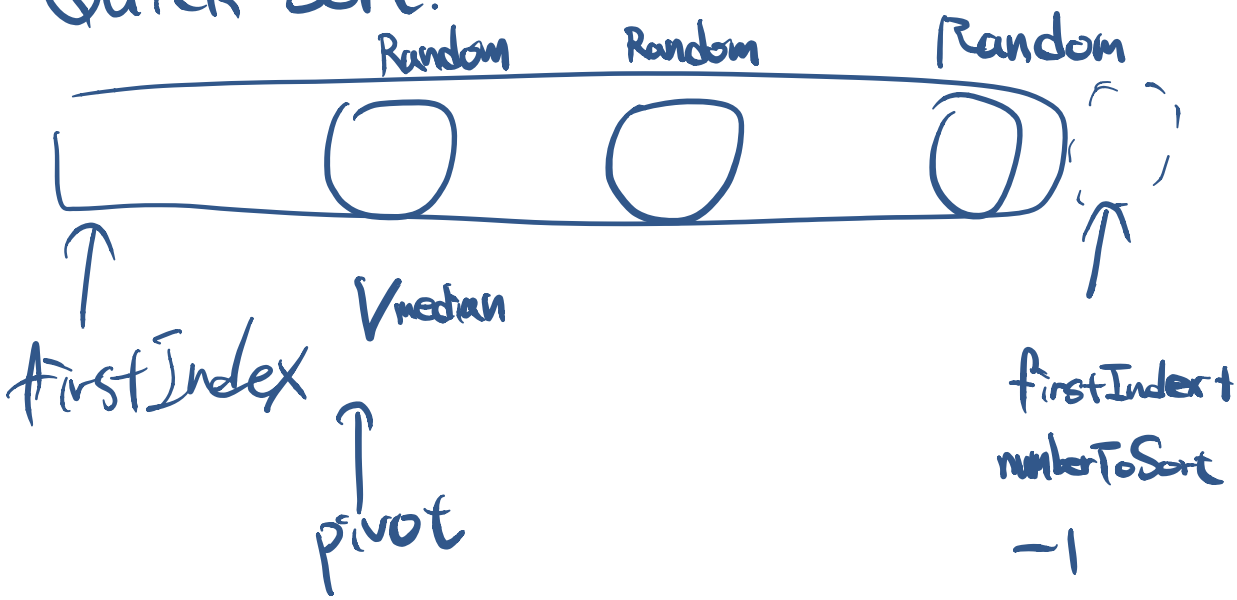
```
// Sort the elements from data[firstIndex] up to
// (but not including) data[firstIndex+n].
```

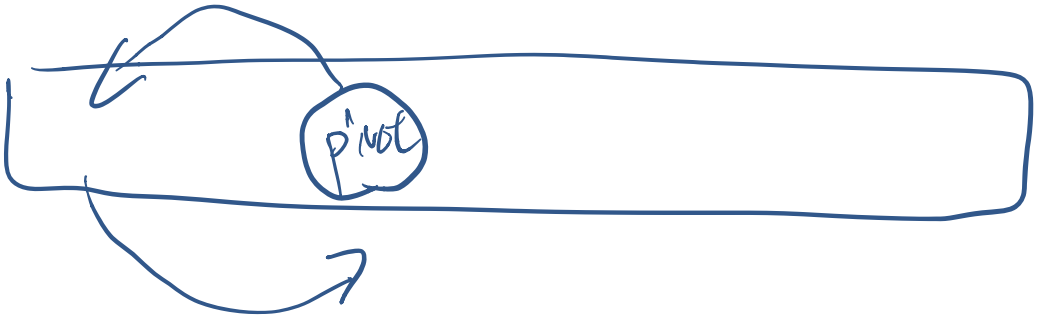
```
public static void mergesort(int[] data, int first, int n)
{
    final int THRESHOLID = 20000;
    int n1; // size of the first half of the array
    int n2; // sizeof the 2nd half of the array
    if ( n > THRESHOLID )
    {
        // Compute sizes of the two halves:
        n1 = n / 2;
        n2 = n - n1;
        // Sort the two ha
```

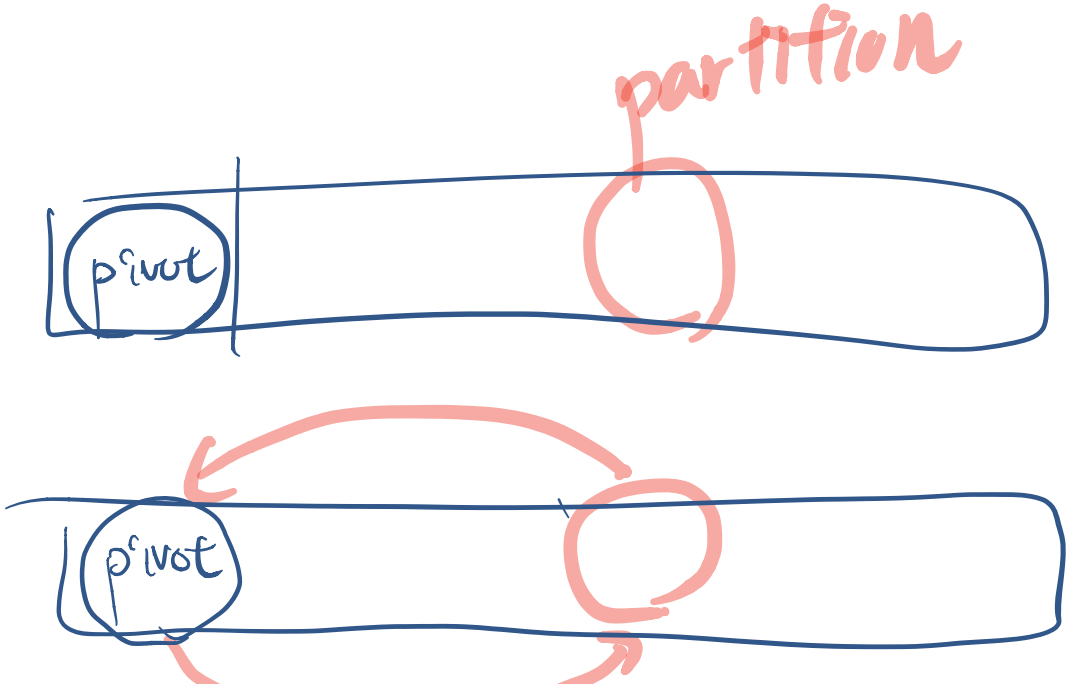
```
}
```

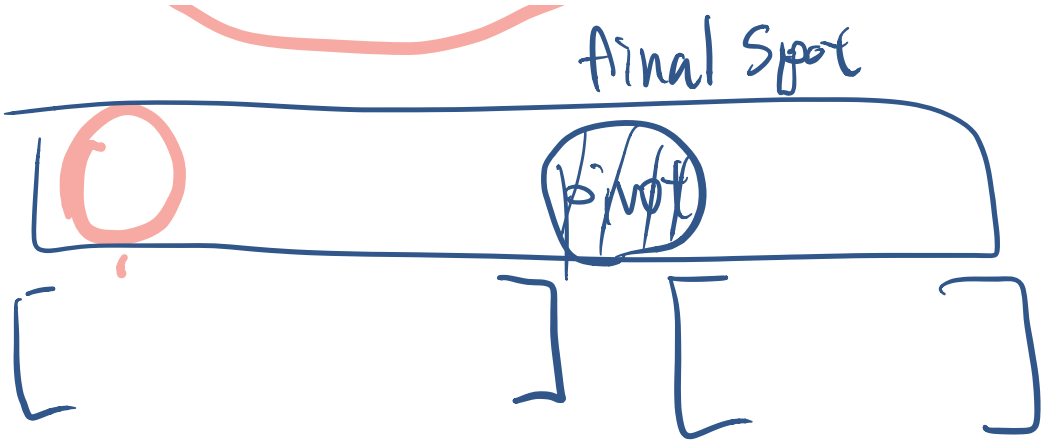

pmd_min
bin
run.sh
right click

Quick Sort.


Random
Random
Random
firstIndex
Vmedian
pivot
firstIndex+
numberToSort
-1


pivot


partition
pivot
pivot


Final Spot
pivot

Heapsort.

Merge Sort worse-case

an interchange sorting algorithm

Array → heap

Complete tree.

pseudo code for **heap sort.**

1. Convert the array of n elements into a heap
2. unsorted = n ; // The number of elements in the unsorted side
3. while (unsorted > 1)

{

// Reduce the unsorted side by 1

unsorted --;

swap data[0] with data [unsorted].

The unsorted side of the array is now a heap with the root out of place.

Do a reheapification downward to turn the unsorted side back into a heap.

```
public static void heapsort(int[]data, int n)
{
    int unsorted; // Size of the unsorted part of the array
    int temp;     // Used during the swapping of two array loc
    makeHeap( data, n );
    while (unsorted > 1)
    {
        unsorted --;

        // swap the largest element with the final
        // element of the unsorted part

        temp = data[0];
        data[0] = data[unsorted];
        data[unsorted] = temp;
        reheapifyDown( data, unsorted );
    }
}
```

Making the Heap

rearrange array ⟶ heap.

private static void makeHeap(int[] data, int n);

→ Complete binary tree is a heap.

FMI) p672 is the more efficient one

pseudo code for makeHeap

```
// Making a heap from an array called data with n elements
for ( i = 1; i < n; ++i )
{
    k = i; // The index of the new element
    while ( data[k] is not yet the the root,
            and data[k] > its parent, and reset
            k to be the index of its parent.)
    {
        Swap data[k] with its parent
        k = index of its parent
    }
// readability
private static int parent( int k )
{
    return (k-1)/2; // assume k > 0
}
```

```
private static void heapifySubtree(
    int[] data,
    int rootOfSubtree,
    int n
)
```

reheapification downward

```
private static void reheapifyDown(int[] data, int n)
```

postcondition: The data values have been rearranged so that the first n elements of data now form a heap.

by continually swapping the out-of-place element with its largest child

```
// Reheapification downward (for a heap) when the root is out of place
int current;    // index of node that's moving down
int bigChildIndex; // index of current's larger child

boolean heapOkay; // will become true when heap is correct
current = 0;
heapOkay = false;
while ((!heapOkay) && (the current node is not a leaf))
{
  Set bigChildIndex to be the index of the larger child
  of the current node. (If there is only one child,
  then bigChildIndex will be set to the index of this
  one child.)
    if (data[current] < data[bigChildIndex])
    {
      Swap data[current] with data[bigChildIndex].
```

```
            current = bigChildIndex;
        }
        else
            heapOkay = true;
    }
```