DIVIDE AND CONQUER

DIVIDE AND CONQUER

Concept

a recursive method of breaking down a problem into unique sub problems to be solved and combined into a solution for the original larger problem

at first this definition may sound like Dynamic Programming where a large problem is solved by resolving smaller sub problems

Dynamic Programming is specifically used to optimize overlapping sub problems by replacing repetitive sub problem calculations with memory storage and lookup

Divide and Conquer divides a large problem into unique sub problems which are not overlapping

Merge Sort

Concept recursively divide array into halved subsets which are merged into sorted order
O(n log n) time complexity

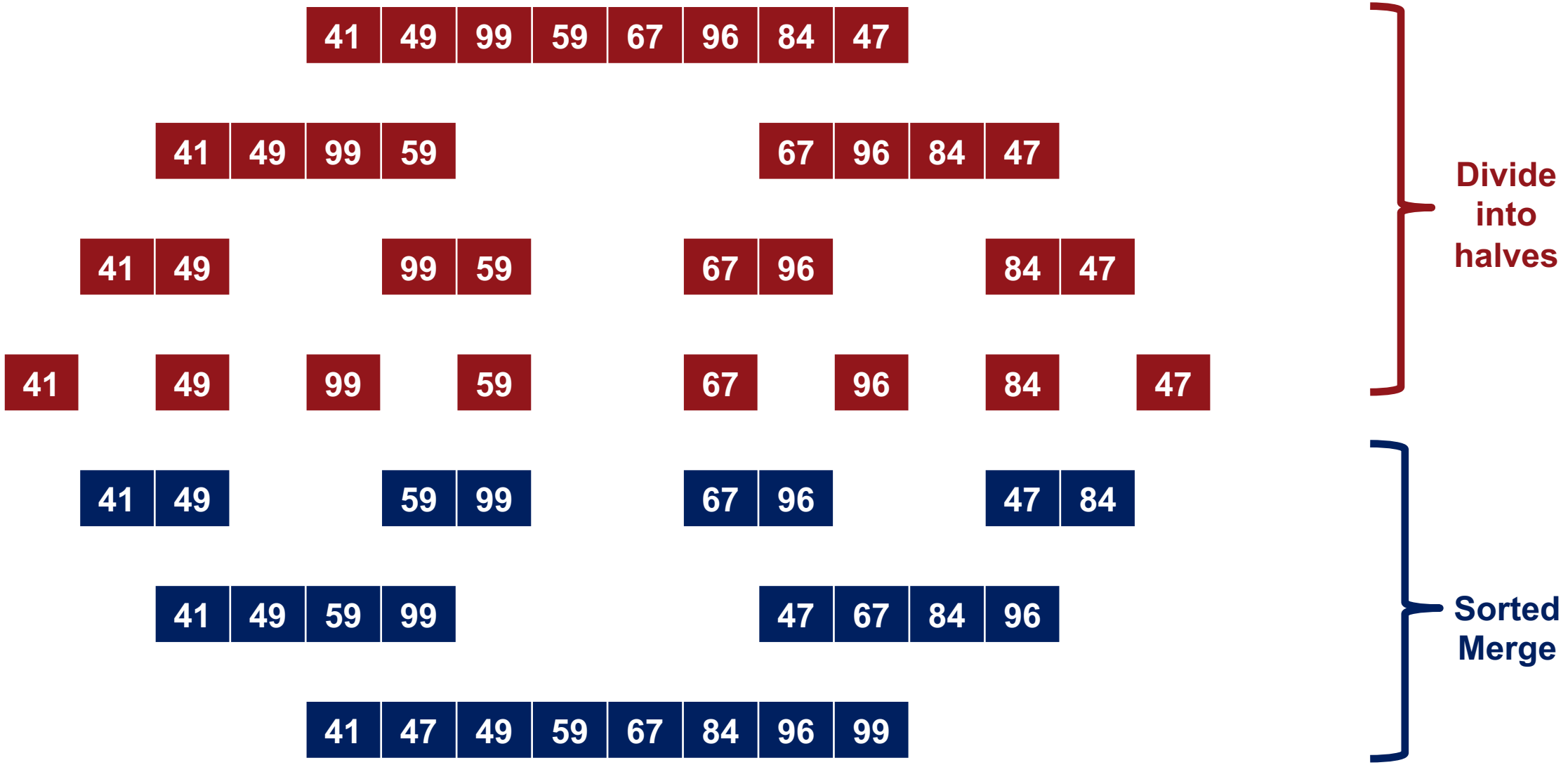
Example

```
void sort(int *a, int start, int end)  {          //  recursively halve array into subsets
     if( start < end) {                           // recurse until out of values
          int middle = (start + end) / 2;         // find middle
          sort(a, start, middle);                 // sort lower half
          sort(a, middle+1, end);                 // upper half
          merge(a, start, middle, end);           // merge
     }
}

void merge(int *a, int start, int middle, int end) {
          // store left and right halves of the array a into new left/right arrays
          // sort the smaller values of left/right arrays into the array a
          // sort any remaining values from left/right arrays into array a
}
```

ex1_merge_sort.cpp

Merge Sort


41 49 99 59 67 96 84 47
41 49 99 59
67 96 84 47
41 49
99 59
67 96
84 47
41
49
99
59
67
96
84
47
41 49
59 99
67 96
47 84
41 49 59 99
47 67 84 96
41 47 49 59 67 84 96 99
Divide into halves
Sorted Merge

Quick Sort

Concept

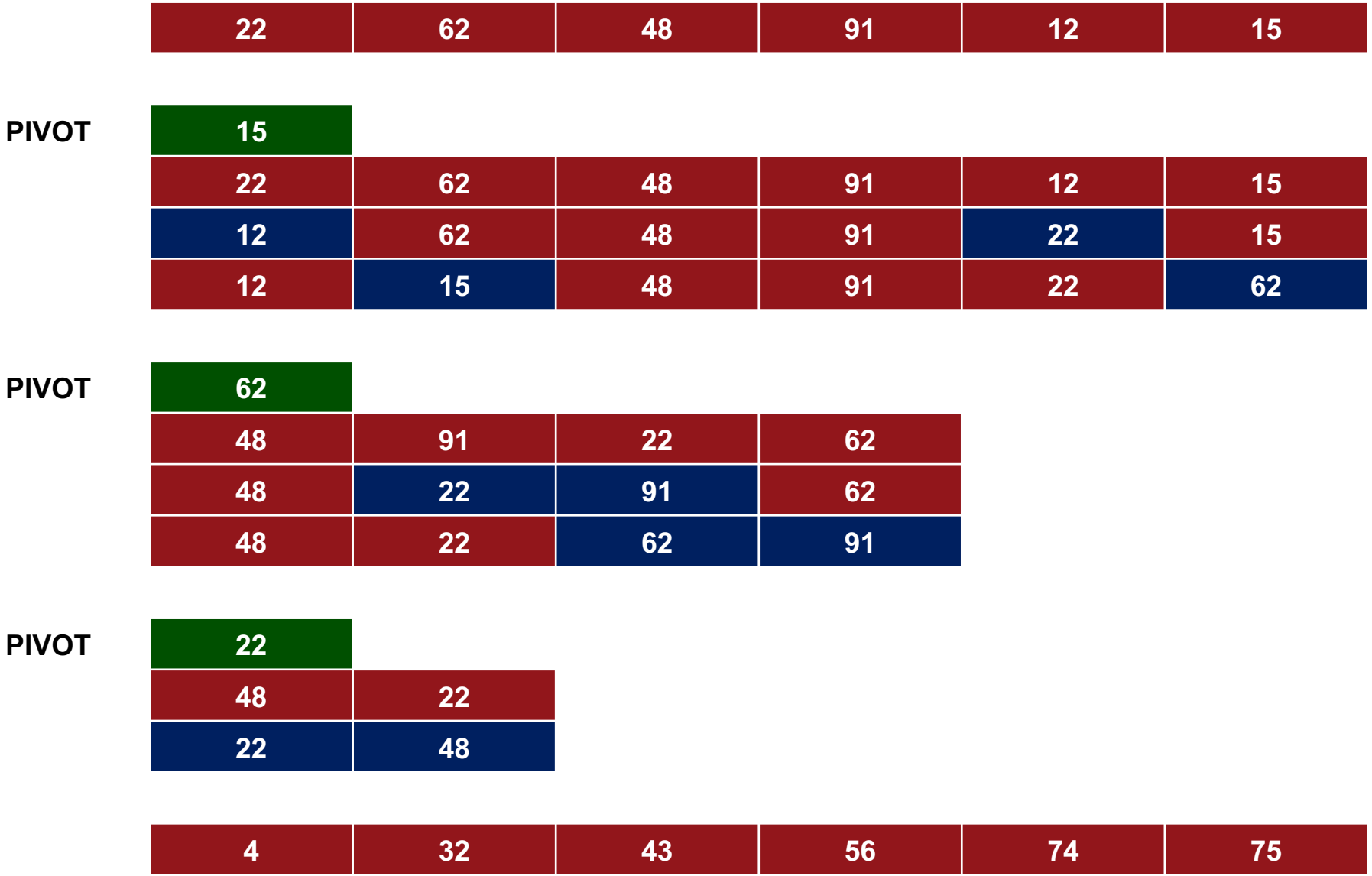
recursively select a pivot value from the array, then partition the array such that:
all values less than the pivot value have a lower index then the pivot value
all values greater than the pivot value have a higher index then the pivot value
$O(n^2)$ time complexity but can be optimized to compete with O(n log n) algorithms

Example

```
void sort(int *a, int low, int high)  {          // recursively partition the array
     if( low < high) {                           // if values remain in the array
          int pivot = partition(a, low, high);   //    find new pivot value
          sort(a, low, pivot-1);                 //    sort partition before pivot
          sort(a, pivot+1, high);                //    sort partition after pivot
     }
}
int partition(int *a, int low, int high) {
     // set pivot to last value
     // place all values relative to the pivot (less than or greater than)
     // swap pivot into its relative position
}
```

ex2_quick_sort.cpp

Quick Sort


22 62 48 91 12 15
PIVOT
15
22 62 48 91 12 15
12 62 48 91 22 15
12 15 48 91 22 62
PIVOT
62
48 91 22 62
48 22 91 62
48 22 62 91
PIVOT
22
48 22
22 48
4 32 43 56 74 75

Merge Sort vs. Quick Sort

Merge Sort

requires more space than quick sort because it creates new arrays

optimal for large data structures stored on disk due to less read operations

Quick Sort

requires less space then merge sort because it sorts in place

optimal for smaller data structures stored in memory

randomized pivot value improves average time complexity
making it potentially faster then merge sort

considered to be the fastest sorting method for most applications