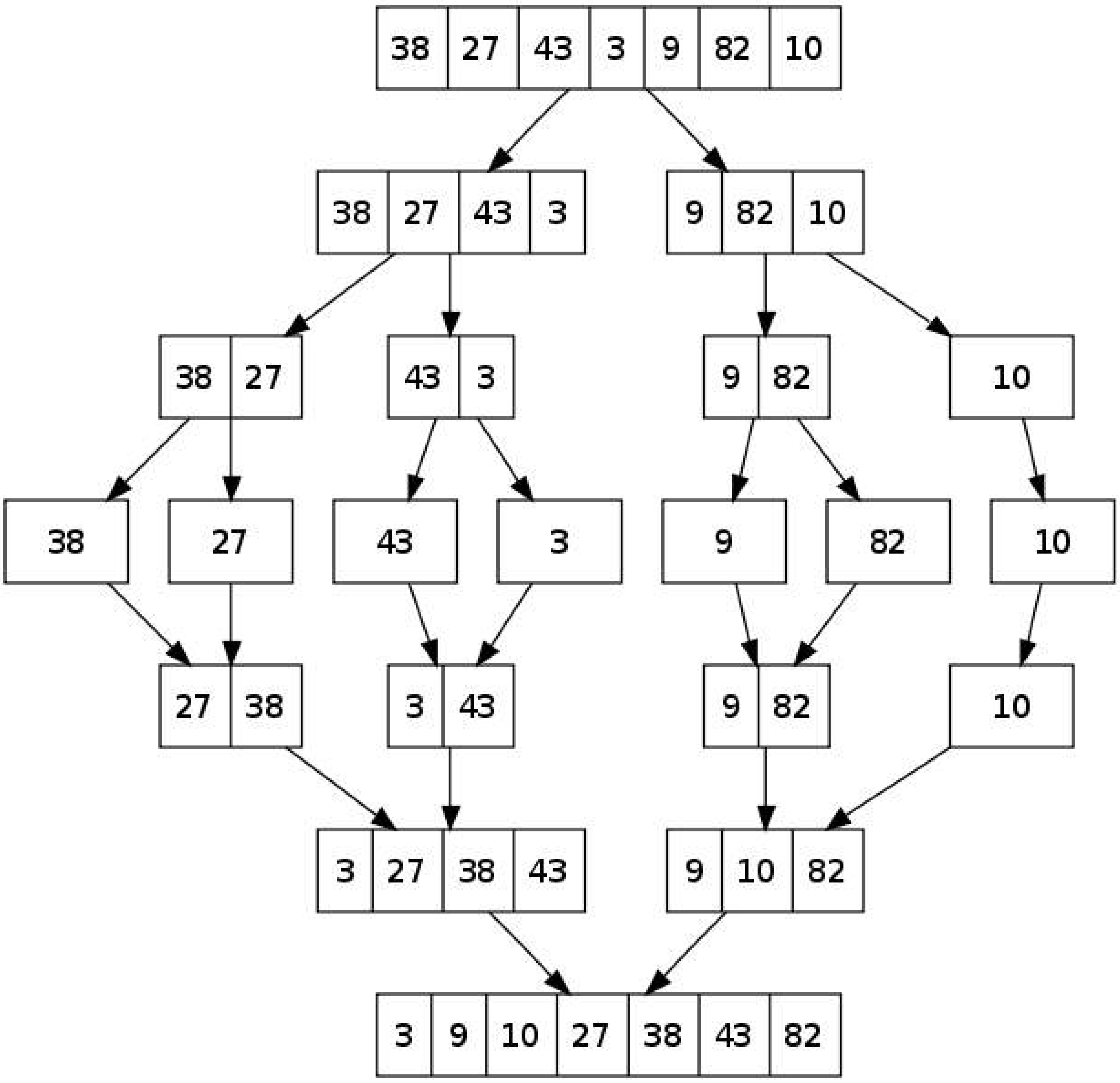
Chapter 30: Merge Sort

Section 30.1: Merge Sort Basics

Merge Sort is a divide-and-conquer algorithm. It divides the input list of length n in half successively until there are n lists of size 1. Then, pairs of lists are merged together with the smaller first element among the pair of lists being added in each step. Through successive merging and through comparison of first elements, the sorted list is built.

An example:



Time Complexity: `T(n) = 2T(n/2) + Θ(n)`

The above recurrence can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is `Θ(nLogn)`. Time complexity of *Merge Sort* is `Θ(nLogn)` in all 3 cases (*worst, average and best*) as merge sort always divides the array in two halves and take linear time to merge two halves.

Auxiliary Space: `O(n)`

Algorithmic Paradigm: Divide and Conquer

Sorting In Place: Not in a typical implementation

Stable: Yes

Section 30.2: Merge Sort Implementation in Go

```go
package main

import "fmt"

func mergeSort(a []int) []int {
    if len(a) < 2 {
        return a
    }
    m := (len(a)) / 2

    f := mergeSort(a[:m])
    s := mergeSort(a[m:])

    return merge(f, s)
}

func merge(f []int, s []int) []int {
    var i, j int
    size := len(f) + len(s)

    a := make([]int, size, size)

    for z := 0; z < size; z++ {
        lenF := len(f)
        lenS := len(s)

        if i > lenF-1 && j <= lenS-1 {
            a[z] = s[j]
            j++
        } else if j > lenS-1 && i <= lenF-1 {
            a[z] = f[i]
            i++
        } else if f[i] < s[j] {
            a[z] = f[i]
            i++
        } else {
            a[z] = s[j]
            j++
        }
    }

    return a
}

func main() {
    a := []int{75, 12, 34, 45, 0, 123, 32, 56, 32, 99, 123, 11, 86, 33}
    fmt.Println(a)
    fmt.Println(mergeSort(a))
}
```

Section 30.3: Merge Sort Implementation in C & C#

C Merge Sort

```
int merge(int arr[],int l,int m,int h)
{
  int arr1[10],arr2[10];  // Two temporary arrays to
  hold the two arrays to be merged
  int n1,n2,i,j,k;
  n1=m-l+1;
  n2=h-m;

  for(i=0; i<n1; i++)
    arr1[i]=arr[l+i];
  for(j=0; j<n2; j++)
    arr2[j]=arr[m+j+1];

  arr1[i]=9999;  // To mark the end of each temporary array
  arr2[j]=9999;

  i=0;
  j=0;
  for(k=l; k<=h; k++) { //process of combining two sorted arrays
    if(arr1[i]<=arr2[j])
      arr[k]=arr1[i++];
    else
      arr[k]=arr2[j++];
  }

  return 0;
}

int merge_sort(int arr[],int low,int high)
{
  int mid;
  if(low<high) {
    mid=(low+high)/2;
    // Divide and Conquer
    merge_sort(arr,low,mid);
    merge_sort(arr,mid+1,high);
    // Combine
    merge(arr,low,mid,high);
  }

  return 0;
}
```

C# Merge Sort

```
public class MergeSort
    {
        static void Merge(int[] input, int l, int m, int r)
        {
            int i, j;
            var n1 = m - l + 1;
            var n2 = r - m;

            var left = new int[n1];
            var right = new int[n2];

            for (i = 0; i < n1; i++)
            {
                left[i] = input[l + i];
            }
            for (j = 0; j < n2; j++)
            {
                right[j] = input[m + j + 1];
            }

            i = 0;
            j = 0;
            var k = l;

            while (i < n1 && j < n2)
            {
                if (left[i] <= right[j])
                {
                    input[k] = left[i];
                    i++;
                }
                else
                {
                    input[k] = right[j];
                    j++;
                }
                k++;
            }

            while (i < n1)
            {
                input[k] = left[i];
                i++;
                k++;
            }

            while (j < n2)
            {
                input[k] = right[j];
                j++;
                k++;
            }
        }

        static void SortMerge(int[] input, int l, int r)
        {
            if (l < r)
            {
                int m = l + (r - l) / 2;
                SortMerge(input, l, m);
                SortMerge(input, m + 1, r);
                Merge(input, l, m, r);
            }
        }

        public static int[] Main(int[] input)
        {
            SortMerge(input, 0, input.Length - 1);
            return input;
        }
    }
```

Section 30.4: Merge Sort Implementation in Java

Below there is the implementation in Java using a generics approach. It is the same algorithm, which is presented above.

```
public interface InPlaceSort<T extends Comparable<T>> {
void sort(final T[] elements); }


public class MergeSort < T extends Comparable < T >> implements InPlaceSort < T > {


@Override
public void sort(T[] elements) {
    T[] arr = (T[]) new Comparable[elements.length];
    sort(elements, arr, 0, elements.length - 1);
}

// We check both our sides and then merge them
private void sort(T[] elements, T[] arr, int low, int high) {
    if (low >= high) return;
    int mid = low + (high - low) / 2;
    sort(elements, arr, low, mid);
    sort(elements, arr, mid + 1, high);
    merge(elements, arr, low, high, mid);
}


private void merge(T[] a, T[] b, int low, int high, int mid) {
    int i = low;
    int j = mid + 1;

    // We select the smallest element of the two. And then we put it into b
    for (int k = low; k <= high; k++) {

        if (i <= mid && j <= high) {
            if (a[i].compareTo(a[j]) >= 0) {
                b[k] = a[j++];
            } else {
                b[k] = a[i++];
            }
        } else if (j > high && i <= mid) {
            b[k] = a[i++];
        } else if (i > mid && j <= high) {
            b[k] = a[j++];
        }
    }

    for (int n = low; n <= high; n++) {
        a[n] = b[n];
    }}}
```

Section 30.5: Merge Sort Implementation in Python

```
def merge(X, Y):
    " merge two sorted lists "
    p1 = p2 = 0
    out = []
    while p1 < len(X) and p2 < len(Y):
        if X[p1] < Y[p2]:
            out.append(X[p1])
            p1 += 1
        else:
            out.append(Y[p2])
            p2 += 1
    out += X[p1:] + Y[p2:]
    return out

def mergeSort(A):
    if len(A) <= 1:
        return A
    if len(A) == 2:
        return sorted(A)

    mid = len(A) / 2
    return merge(mergeSort(A[:mid]), mergeSort(A[mid:]))

if __name__ == "__main__":
    # Generate 20 random numbers and sort them
    A = [randint(1, 100) for i in xrange(20)]
    print mergeSort(A)
```

Section 30.6: Bottoms-up Java Implementation

```
public class MergeSortBU {
    private static Integer[] array = { 4, 3, 1, 8, 9, 15, 20, 2, 5, 6, 30, 70,
60,80,0,9,67,54,51,52,24,54,7 };

    public MergeSortBU() {
    }

    private static void merge(Comparable[] arrayToSort, Comparable[] aux, int lo,int mid, int hi) {

        for (int index = 0; index < arrayToSort.length; index++) {
            aux[index] = arrayToSort[index];
        }

        int i = lo;
        int j = mid + 1;
        for (int k = lo; k <= hi; k++) {
            if (i > mid)
                arrayToSort[k] = aux[j++];
            else if (j > hi)
                arrayToSort[k] = aux[i++];
            else if (isLess(aux[i], aux[j])) {
                arrayToSort[k] = aux[i++];
            } else {
                arrayToSort[k] = aux[j++];
            }

        }
    }

    public static void sort(Comparable[] arrayToSort, Comparable[] aux, int lo, int hi) {
        int N = arrayToSort.length;
        for (int sz = 1; sz < N; sz = sz + sz) {
            for (int low = 0; low < N; low = low + sz + sz) {
                System.out.println("Size:"+ sz);
                merge(arrayToSort, aux, low, low + sz -1 ,Math.min(low + sz + sz - 1, N - 1));
                print(arrayToSort);
            }
        }

    }

    public static boolean isLess(Comparable a, Comparable b) {
        return a.compareTo(b) <= 0;
```

```java
    }

    private static void print(Comparable[] array)
{http://stackoverflow.com/documentation/algorithm/5732/merge-sort#
        StringBuffer buffer = new
StringBuffer();http://stackoverflow.com/documentation/algorithm/5732/merge-sort#
        for (Comparable value : array) {
            buffer.append(value);
            buffer.append(' ');
        }
        System.out.println(buffer);
    }

    public static void main(String[] args) {
        Comparable[] aux = new Comparable[array.length];
        print(array);
        MergeSortBU.sort(array, aux, 0, array.length - 1);
    }
}
```