Lecture 3: Heap Sort and Quick Sort

- Learn how performance of Merge sort can be further improved by using Heap sort and Quick Sort

---

Heap Sort

- Use "heap" data structure to manage information
- Makes also an efficient priority queue (more later in the semester)

Binary Heap

pseudo-code for binary heap

```
Parent(i)
    return (i-1)/2
Left(i)
    return 2i+1
Right(i)
    return 2(i+1)
```

Max-heap

- A [Parent(i)] >= A[i]
- Largest value stored at root
- Subtree rooted at node contains no values larger than value of node itself
- Used for heapsort

Min-heap

- A [Parent(i)] <= A[i]
- Smallest value stored at root
- Subtree rooted at node contains no values smaller than value of node itself

---

Max Heapify

```
l<-LEft(i)
r->RIGHT(i)
if I<= heap-size[A] and A[l] < A[i]
```

```
    largest = l
else
    largest = r
if largest != i
    then exchange A[i] <-> largest
    MAX-HEAPIFY(A, largest)

#build heap
heap-size[A] <- length[A]
for I <- ⌊length[A]/2⌋ downto 1
do MAX-HEAPIFY(A, i)
```

Heap Sort

```
MAX-HEAPIFY(A)
for i←length[A] downto 2
    do exchange A[1] ⟷ A[i]
        heap-size[A] <- heap-size[A] – 1
    MAX-HEAPIFY(A, 1)
```

Heap Sort Analysis

- Heapsort takes time O(nlgn)
- BUILD-MAX-HEAP takes time O(n)
- MAX-HEAPIFY takes O(lgn)

Quick Sort

- Uses divide and conquer like Merge Sort
- No ADDITIONAL storage usage => overcomes merge sort weakness
- Trade off: performance diminished if list can be divided in half

---

Pivot Value

- **1) Select Pivot Value**
    - Simplify first item in list
    - Assist with splitting list
    - **Split point** used to divide list of subsequent calls

    - Set leftmark and rightmark
    - Move items to "right" side of pivot value
    - Converge split point
- **3) Exchange Pivot**
    - Stop when rightmark <= leftmark
    - Move items to the "right" side of pivot value
    - Converge to split point
    - Quick sort for left and right half

QuickSort Code

```
def quickSort(alist):
    quickSortHelper(alist,0,len(alist)-1)

def quickSortHelper(alist,first,last):
    if first<last:

        splitpoint = partition(alist,first,last)

        quickSortHelper(alist,first,splitpoint-1)
        quickSortHelper(alist,splitpoint+1,last)

def partition(alist,first,last):
    pivotvalue = alist[first]
    leftmark = first+1
    rightmark = last
    done = False
    while not done:
        while leftmark <= rightmark and alist[leftmark] <= pivotvalue:
            leftmark = leftmark + 1
        while alist[rightmark] >= pivotvalue and rightmark >= leftmark:
            rightmark = rightmark -1
    if rightmark < leftmark:
        done = True
    else:
        temp = alist[leftmark]
        alist[leftmark] = alist[rightmark]
        alist[rightmark] = temp
    temp = alist[first]
    alist[first] = alist[rightmark]
    alist[rightmark] = temp
    return rightmark

alist = [54,26,93,17,77,31,44,55,20]
```