BLG 335E – Analysis of Algorithms I
Fall 2013, Recitation 3
06.11.2013

R.A. Atakan Aral
aralat@itu.edu.tr – Research Lab 1
R.A. Doğan Altan
daltan@itu.edu.tr – Research Lab 3

Preliminary Information

- Although **MAX-HEAPIFY** costs $O(\lg n)$ time and there are $O(n)$ calls to it, still we can find a tighter bound than $O(n \lg n)$ for:

```
BUILD-MAX-HEAP(A)
heap-size[A] ← length[A]
for i ← ⌊length[A]/2⌋ downto 1
      do MAX-HEAPIFY(A, i)
```

- Because, cost of **MAX-HEAPIFY** depends on the height of the node in the tree, and and the heights of most nodes are small.

Exercise 1

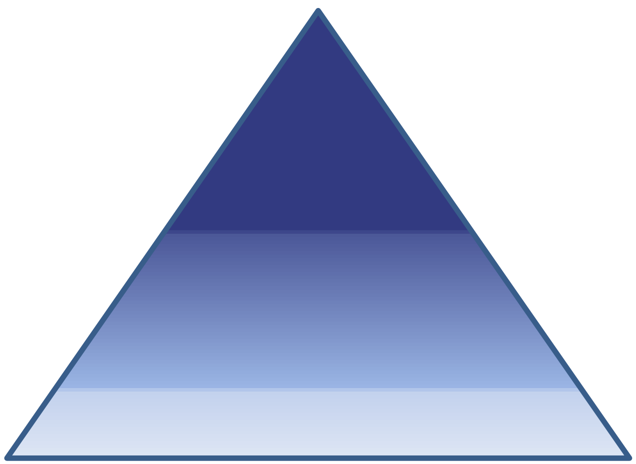
- Using a similar reasoning, can we find a tighter bound than *O(n lg n)* for **Heapsort**?

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i ← length[A] downto 2
    do exchange A[1] ↔ A[i]
       heap-size[A] ← heap-size[A] − 1
       MAX-HEAPIFY(A, 1)
```

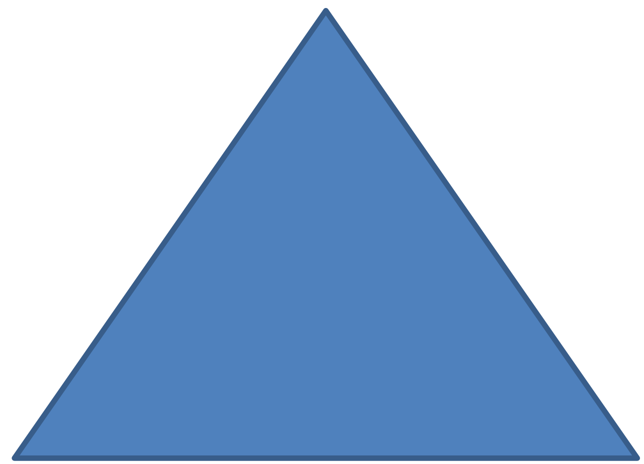
Exercise 1 – Solution

```
BUILD-MAX-HEAP(A)
heap-size[A] ← length[A]
for i ← ⌊length[A]/2⌋ downto 1
        do MAX-HEAPIFY(A, i)
```

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i ← length[A] downto 2
        do exchange A[1] ↔ A[i]
            heap-size[A] ← heap-size[A] − 1
            MAX-HEAPIFY(A, 1)
```



$O(n)$



$O(n \log n)$

Exercise 2

- A '**d-ary**' heap is like a binary heap, but non-leaf nodes have d children instead of 2 children.
- How would you represent a d-ary heap in an array?
- Verify that:

  d-ary-parent(d-ary-child(i, j)) = i

Exercise 2 – Solution

binary-parent(i)

$\quad$ return $\lfloor i / 2 \rfloor$

binary-child(i,j)

$\quad$ return $2i - 1 + j$

d-ary-parent(i)

$\quad$ return $\lfloor ((i-2)/d) + 1 \rfloor$

d-ary-child(i,j)

$\quad$ return $d(i-1)+1+j$

- Root's d children are kept in A[2]~A[d+1]
- Their children are kept in A[d+2]~ A[d^2+d+1] and so on.

Exercise 3

- Banks often record transactions on an account in order of the **times of the transactions**.
- But many people like to receive their bank statements with checks listed in order by **check number**.
- Banks need to convert time-of-transaction ordering to check-number ordering.
- Insertion Sort vs. Quick Sort

Exercise 3 – Solution

- People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch.
- The problem is therefore the problem of sorting **almost-sorted** input.
- For **Quick Sort**, best and average case time complexity are the same ($O(n \log n)$).
- For **Insertion Sort**, the best case is $O(n)$ and the average case is $O(n+d)$.

Exercise 4

- Illustrate the operation of **Counting-Sort** on the array below.

  A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]

1. Max{A[i]}=6
2.

0	0	0	0	0	0	0

3.

2	2	2	2	1	0	2

4.

2	4	6	8	9	9	11

Exercise 4

6	0	2	0	1	3	4	6	1	3	2

5.

1	2	3	4	5	6	7	8	9	10	11
					2					
					2		3			
			1		2		3			
			1		2		3			6
			1		2		3	4		6
			1		2	3	3	4		6
		1	1		2	3	3	4		6
	0	1	1		2	3	3	4		6
	0	1	1	2	2	3	3	4		6
0	0	1	1	2	2	3	3	4		6
0	**0**	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**6**	**6**

0	1	2	3	4	5	6
2	4	6	8	9	9	11
2	4	5	8	9	9	11
2	4	5	7	9	9	11
2	3	5	7	9	9	11
2	3	5	7	9	9	10
2	3	5	7	8	9	10
2	3	5	6	8	9	10
2	2	5	6	8	9	10
1	2	5	6	8	9	10
1	2	4	6	8	9	10
0	2	4	6	8	9	10
0	2	4	6	8	9	9

Exercise 5

- Illustrate the operation of **Radix-Sort** on the following list of English words:

  COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

Exercise 5 – Solution

COW
DOG
SEA
RUG
ROW
MOB
BOX
TAB

BAR
EAR
TAR
DIG
BIG
TEA
NOW
FOX

Exercise 5 – Solution

SE**A**	BA**R**
TE**A**	EA**R**
MO**B**	TA**R**
TA**B**	FO**X**
DO**G**	BO**X**
RU**G**	CO**W**
DI**G**	RO**W**
BI**G**	NO**W**

Exercise 5 – Solution

T**A**B
B**A**R
E**A**R
T**A**R
S**E**A
T**E**A
D**I**G
B**I**G

M**O**B
D**O**G
F**O**X
B**O**X
C**O**W
R**O**W
N**O**W
R**U**G

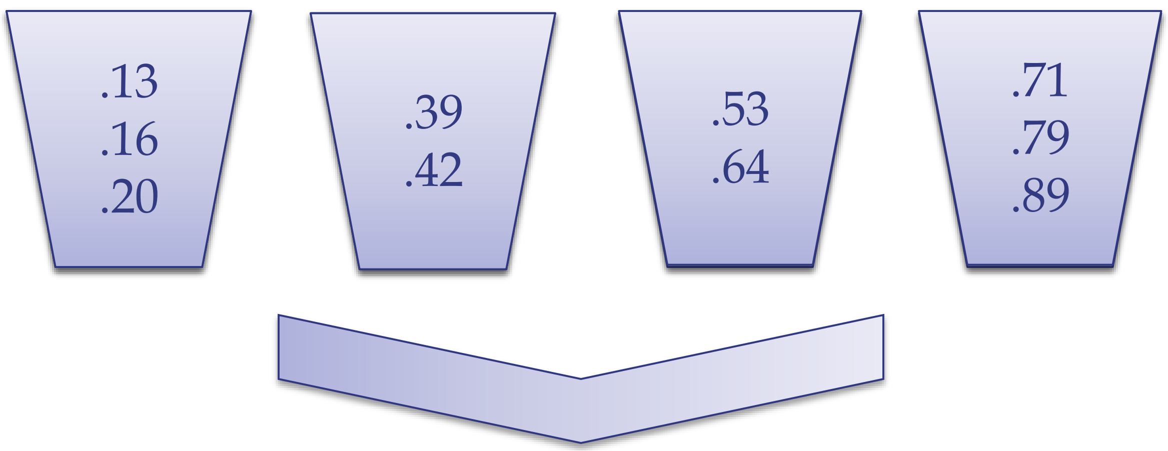
Exercise 5 – Solution

BAR
BIG
BOX
COW
DIG
DOG
EAR
FOX
MOB
NOW
ROW
RUG
SEA
TAB
TAR
TEA

Exercise 6

- Illustrate the operation of **Bucket-Sort** on the array below.

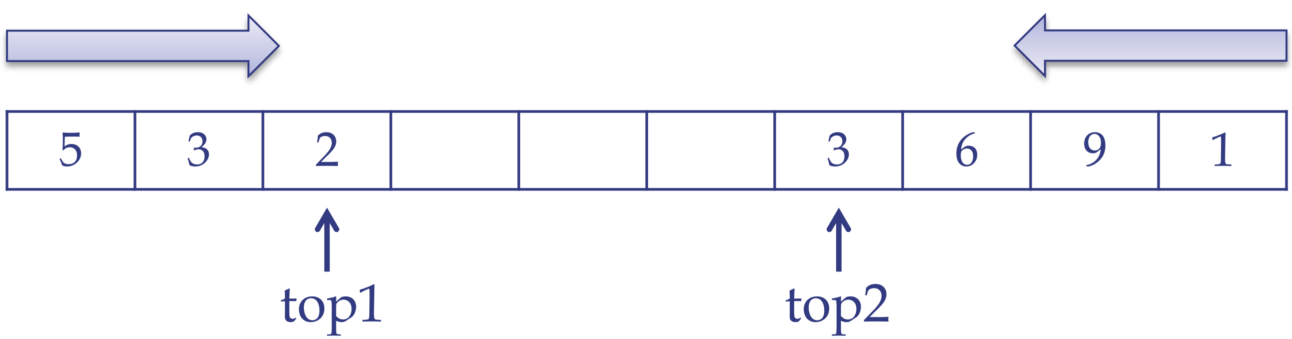
A = [.79, .13, .16, .64, .39, .20, .89, .53, .71, .42]



A = [.13, .16, .20, .39, .42, .53, .64, .71, .79, .89]

Exercise 7

- Explain how to implement **two stacks** in **one array** A[1…n] in such a way that
  - neither stack overflows unless the total number of elements in both stacks is n.
  - the PUSH and POP operations should run in O(1) time.



Extra Exercises

- Implement a **queue** by a singly linked list **L**. The operations **Enqueue** and **Dequeue** should still take **O(1)** time.

- Write an **O(n)-time** procedure that, given an n-node **binary tree**, prints out the key of each node in the tree.
  a) Recursively
  b) Non-recursively using a **stack**