```
case 1, 1':
    z.p.color = **BLACK**
    y.color = **BLACK**
    z.p.p.color = **RED**
    z = z.p.p
case 2:
    z = z.p
    LEFT_ROTATE (T, z)
  case 2'
case 3:
    z.p.color = **BLACK**
    z.p.p.color = **RED**
    RIGHT_ROTATION (T, z.p.p)
  case 3'
```

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

- The MAX-HEAPIFY procedure, which runs in $O(\lg n)$ time, is the key to maintaining the max-heap property.
- The BUILD-MAX-HEAP procedure, which runs in linear time, produces a max-heap from an unordered input array.
- The HEAPSORT procedure, which runs in $O(n \lg n)$ time, sorts an array in place.
- The MAX-HEAP-INSERT, HEAP-EXTRACT-MAX, HEAP-INCREASE-KEY, and HEAP-MAXIMUM procedures, which run in $O(\lg n)$ time, allow the heap data structure to implement a priority queue.

```
RB-INSERT-FIXUP(T, z)
while z.p.color == RED
    if z.p == z.p.p.left
        y = z.p.p.right
        if y.color == RED
            z.p.color = BLACK                    // case 1
            y.color = BLACK                      // case 1
            z.p.p.color = RED                    // case 1
            z = z.p.p                            // case 1
        else if z == z.p.right
                z = z.p                          // case 2
                LEFT-ROTATE(T, z)                // case 2
            z.p.color = BLACK                    // case 3
            z.p.p.color = RED                    // case 3
            RIGHT-ROTATE(T, z.p.p)               // case 3
    else (same as then clause
            with "right" and "left" exchanged)
```

Provide a tight bound for the running time of finding the **biggest** element in a binary **min-heap** with n elements.

```
findBiggestElement (A, n)                 Cost    # of executions
    biggest = A[n]                        C1      1
        for i = n/2 + 1 to (n – 1)        C2      n – 1 – n/2 – 1 + 1 = n/2 – 1
            if A[i] ≥ biggest             C3      1
                biggest = A[i]            C4      1
    return biggest                        C5      1
```

$T(n) = C*1 + C2*(\frac{n}{2} – 1) + C3*1 + C4*1 + C5*1 = O(\frac{n}{2}) + C = O(n)$

Provide a tight bound for the running time of finding the **smallest** element in a binary **max-heap** with n elements.

```
findBiggestElement (A, n)                 Cost    # of executions
    smallest = A[n]                       C1      1
        for i = n/2 + 1 to (n – 1)        C2      n – 1 – n/2 – 1 + 1 = n/2 – 1
            if A[i] < smallest            C3      1
                smallest = A[i]           C4      1
    return smallest                       C5      1
```

$T(n) = C*1 + C2*(\frac{n}{2} – 1) + C3*1 + C4*1 + C5*1 = O(\frac{n}{2}) + C = O(n)$

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

The operation HEAP-DELETE (A, i) deletes the item in node i from heap A. Give an implementation of HEAP-DELETE that runs in O(*log* n) time for an *n*-element max-heap.

```
(1) Pseudocode                                                  Cost    # of executions
    HEAP-DELETE (A, A.heap-size, i)
    (1)  A[i] = A[A.heap-size]                                  C1      1
             /* overwrites the value of A[i] by the value of the smallest leaf */
    (2)  A.heap-size -= 1                                       C2      1
    (3)  while (i > 1 and parent of i < A[i])                   C3      1
    (4)      swap (A[parent(i)]) with A[i]                      C4      1
    (5)      i = parent(i)                                      C5      1
    (6)  MAX-HEAPIFY(A, i)                                      C6      1
```

(2) correctness justification

Done in the answer (1)

(2) provide an upper bound of your procedure and give an explanation

Explanation: as of A[i] has been removed and replaced by the smallest leaf, A[i] must be iteratively compared with its parent then swap if the parent is ≤ A[i] to ensure A[parent(i)] ≥ A[i] (line 3, 4, 5).

In case of A[i] is small, thus it recursively swaps with its child and traverses the longest path until it becomes a leaf.

$T(n) = C1*1 + C2*1 + C3*\sum_2^h 1 + C4*\sum_2^h 1 + C5*\sum_2^h 1 + C6*1$
$= C1 + C2 + (C3 + C4 + C5)*(h-2+1) + C6$
$= C*(h-1) + D$ (C and D are positive constants)
$= O(h) + D$ (provided h = *lg* n)
$= O(lg\ n)$ = > proved

```
RB_INSERT_FIXUP (T, z)
    while (z.p is **RED**) and (z ≠ T.root)
        if z.p is a **LEFT** child
            y is an **RIGHT** uncle
            if y is **RED**
                <case 1>
            else
                if z is a **RIGHT** child
                    <case 2> then continue to <case 3>
                <case 3>
        else    // z.p is a **RIGHT** child
            // same as above but **RIGHT** ⇔ **LEFT**
            y is an **LEFT** uncle
            if y is **RED**
                <case 1'>
            else
                if z is a **LEFT** child
                    <case 2'> then continue to <case 3'>
                <case 3'>
    T.root = **BLACK**
```

Root and NIL leaf: BLK
R must have 2 BLK children
Maintain BLK-height

Running time of Quicksort
$T(n) = \Theta(n)+T(q-1)+T(n-q)$
$= T(n-1)+cn = \Theta(n^2)$

$T(n)= T(n/2)+c$
$\therefore \Theta(lgn)$
$T(n)=4T(n/2)+cn$
$\therefore \Theta(n^2-n)=\Theta(n^2)$
$T(n)=T(\alpha n)+T((1-\alpha)n)+cn$
$\therefore O(nlgn)$

```
HEAPSORT(A)    T(n)=O(nlgn)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size – 1
    MAX-HEAPIFY(A, 1)
```

Slower than Quicksort

Maintaining the heap property T(n) = O(lg n)

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

T(n) = const time + time of M-H =T(1)+O(lgn) = O(lgn)

```
HEAP-EXTRACT-MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-size – 1
MAX-HEAPIFY(A, 1)
return max
```

$T(n) = a.T(n/b) + f(n)$
$a \geq 1, b>1, f(n)>0, \neq 0$ as $n \to \infty$
Compare $n^{\log_b a}$ vs. $f(n)$
Case 1: $n^{\log_b a} > f(n)$
$T(n) = \Theta(n^{\log_b a})$
Case 2: $n^{\log_b a} = f(n)$
$T(n) = \Theta(n^{\log_b a} * lgn)$
Case 3: $n^{\log_b a} < f(n)$
$T(n) = \Theta(f(n))$

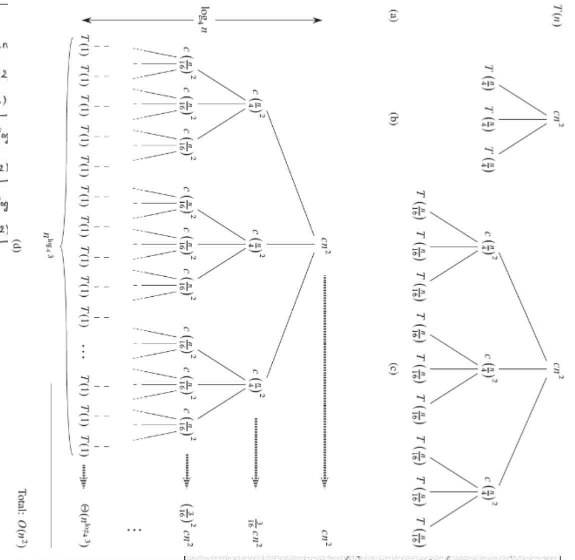
$T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(nlgn)$
Upper bound:
Guess $T(n) = O(nlgn)$
$T(n) \leq dnlgn$ (d: pos. const.)
$T(\frac{n}{2}) \leq d\frac{n}{2}lg\frac{n}{2}$
Substitute:
$T(n) \leq 2d(\frac{n}{2})lg\frac{n}{2} + cn$
$= dnlg\frac{n}{2}+cn$
$=dn(lgn – 1) + cn$
$\leq dnlgn$ if $(-dn+cn \leq 0)$
$=> c \leq d$
$\therefore T(n) = O(nlgn)$

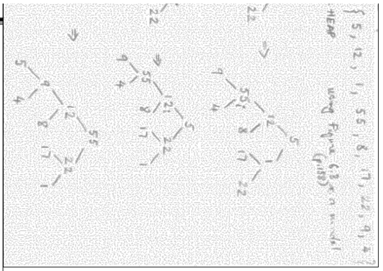
```
MAX-HEAP-INSERT(A, key)
A.heap-size = A.heap-size + 1
A[A.heap-size] = –∞
HEAP-INCREASE-KEY(A, A.heap-size, key)
```

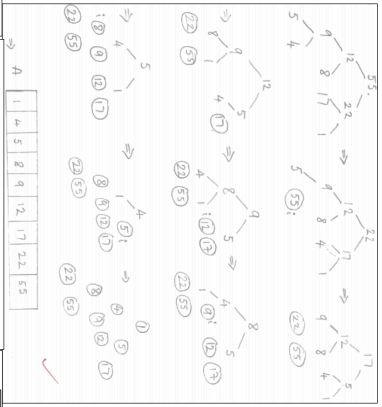
$T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(nlgn)$
Lower bound:
Guess $T(n) = \Omega(nlgn)$
$T(n) \geq dnlgn$ (d: pos. const.)
$T(\frac{n}{2}) \geq d\frac{n}{2}lg\frac{n}{2}$
Substitute:
$T(n) \geq 2d(\frac{n}{2})lg\frac{n}{2} + cn$
$= dnlg\frac{n}{2}+cn$
$=dn(lgn – 1) + cn$
$\geq dnlgn$ if $(-dn+cn \geq 0)$
$=> c \geq d$
$\therefore T(n) = \Omega(nlgn)$
$\therefore T(n) = \Theta(nlgn)$

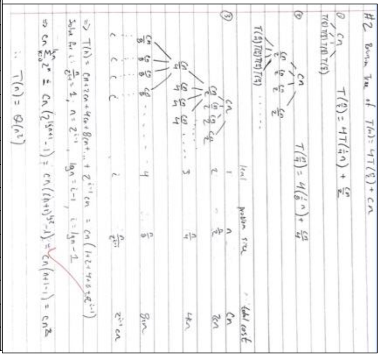
$T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$
$= 8T(\frac{n}{2}) + cn^2$
Lower bound:
Guess: $T(n) = \Omega(n^3)$
$T(n) \geq dn^3$ $(d > 0)$
$T(\frac{n}{2}) \geq d(\frac{n}{2})^3$
Substitute:
$T(n) \geq 8(dn^3/8) +cn^2= dn^3+cn^2$
$\geq dn^3$ if $cn^2 \geq 0$
=> always happens
$\therefore T(n) = \Omega(n^3)$ $\therefore T(n) = \Theta(n^3)$
$\mathbf{T(n) = 4T(\frac{n}{4})+n => O(nlgn)}$

$T(n) = 8T(\frac{n}{2}) + \Theta(n^2) = 8T(\frac{n}{2}) + cn^2$
Upper bound:
Guess: $T(n) = O(n^3)$
$T(n) \leq dn^3$ $(d > 0)$
$T(\frac{n}{2}) \leq d(\frac{n}{2})^3$
Substitute:
$T(n) \leq 8(dn^3/8) +cn^2 = dn^3+cn^2$
$\leq dn^3$ if $cn^2 \leq 0$ => no c and n exist
New guess (–a lower-order term)
$T(n) \leq dn^3 – d'n^2$ (d, d'>0)
$T(\frac{n}{2}) \leq d(\frac{n}{2})^3 – d'(\frac{n}{2})^2$
$= dn^3/8 – d'n^2/4$
Substitute: $T(n) \leq 8T(\frac{n}{2}) + cn^2$
$\leq 8(dn^3/8 – d'n^2/4) + cn^2$
$= dn^3 – 2d'n^2 + cn^2$
$\leq dn^3 – d'n^2$ if $(–d'n^2 + cn^2) \leq 0$
$=> c \leq d'$ $\therefore T(n) = O(n^3)$








#2 Recursion Tree of $T(n) = 4T(\frac{n}{2}) + cn$

① cn $T(\frac{n}{2}) = 4T(\frac{1}{4}n) + \frac{cn}{2}$

$T(\frac{n}{4})T(\frac{n}{4})T(\frac{n}{4})T(\frac{n}{4})$

② cn $T(\frac{n}{4}) = 4(\frac{1}{8}n) + \frac{cn}{4}$

③

level	problem size	total cost
1	n	cn
2	$\frac{n}{2}$	$2cn$
3	$\frac{n}{4}$	$4cn$
4	$\frac{n}{8}$	$8cn$
i	$\frac{n}{2^{i-1}}$	$2^{i-1}cn$

$\Rightarrow T(n) = cn + 2cn + 4cn + 8cn + \ldots + 2^{i-1}cn = cn(1+2+4+8+\ldots+2^{i-1})$

Solve for i: $\frac{n}{2^{i-1}} = 1$, $n = 2^{i-1}$, $\lg n = i-1$, $i = \lg n - 1$

$\Rightarrow cn\sum_{k=0}^{\lg n} 2^k = cn(2^{\lg n+1} - 1) = cn((n+1)^{\lg 2} - 1) = cn(n+1-1) = cn^2$

$\therefore T(n) = \Theta(n^2)$

20 2/ Use the Recursion Tree

$T(n) = T(\frac{n}{3}) + T(\frac{n}{3}) + 2^{(\log_3 n)}$ (.n

$= 2T(\frac{n}{3}) + n^{(\log_3 2)}$

Given $n = 3^i$, $T(\frac{n}{3}) = 2.T(\frac{n}{9}) + (\frac{n}{3})^{(\log_3 2}$

$= 2T(\frac{n}{9}) + \frac{n^{(\log_3 2)}}{2}$

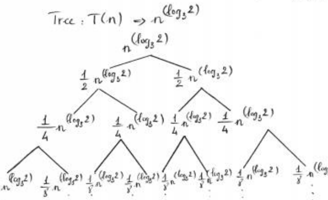
$T(\frac{n}{9}) = 2T(\frac{n}{27}) + (\frac{n}{9})^{(\log}$

$= 2T(\frac{n}{27}) + \frac{n^{(\log_3 2)}}{4}$

$T(\frac{n}{27}) = 2T(\frac{n}{81}) + (\frac{n}{27})^{(\log}$

$= 2T(\frac{n}{81}) + \frac{n^{(\log_3 2)}}{8}$

Tree : $T(n) \Rightarrow n^{(\log_3 2)}$



Level	T(n)	Cost	Factor	
1	$T(n)$	$n^{(\log_3 2)}$	1	$n^{(\log_3 2)}$
2	$T(\frac{n}{3})$	$\frac{1}{2}.n^{(\log_3 2)}$	2	$n^{(\log_3 2)}$
3	$T(\frac{n}{9})$	$\frac{1}{4}.n^{(\log_3 2)}$	4	$n^{(\log_3 2)}$
⋮	⋮	⋮	⋮	⋮
i	$T(\frac{n}{3^{i-1}})$	$\frac{1}{2^{i+1}}.n^{(\log_3 2)}$	2^{i-1}	$n^{(\log_3 2)}$

$T(n) = (\text{\# of level} \times n^{(\log_3 2)})$

$= (\lg n + 1) \times n^{(\log_3 2)}$

$= \lg n . n^{(\log_3 2)} + n^{(\log_3 2)}$

$\boxed{T(n) = \theta(\lg n . n^{(\log_3 2)})}$

$\frac{n}{2^{i-1}} = 1 \Rightarrow 2^{i-1} = n$

$\Rightarrow i - 1 = \lg n$

$\Rightarrow i = \lg n + 1$

3/ Uniform Random Permutation

Exercise 5.3.3

No.

Because in each n iteration, the algorithm chooses the index i independently and uniformly at random {1….n}

Therefore, there are n^n different possible sequences, each has $\Pr = \frac{1}{n^n}$.

There are n! distinct permutations. To get a uniform distribution over permutations, each has Pr

$= \frac{1}{n!}$.

This we have $\frac{k}{n^n} = \frac{1}{n!} \Leftrightarrow n^n$= kn! (k is an integer), thus not possible because n^n is not divisible by

n! (except for n=1 or n=2)

Counting sort <6, 0, 2, 6, 0, 8>

A=	6	0	2	6	0	8		C=	2	0	1	0	0	0	2	0	1
idx	1	2	3	4	5	6		idx	0	1	2	3	4	5	6	7	8
B=	-	-	-	-	-	8		C=	2	2	3	3	3	3	5	5	***6₋₁***
	1	2	3	4	5	6			0	1	2	3	4	5	6	7	8
B=	-	0	-	-	-	8		C=	***2₋₁***	2	3	3	3	3	5	5	5
	1	2	3	4	5	6			0	1	2	3	4	5	6	7	8
B=	-	0	-	-	6	8		C=	1	2	3	3	3	3	***5₋₁***	5	5
	1	2	3	4	5	6			0	1	2	3	4	5	6	7	8
B=	-	0	2	-	6	8		C=	1	2	***3₋₁***	3	3	3	4	5	5
	1	2	3	4	5	6			0	1	2	3	4	5	6	7	8
B=	0	0	2	-	6	8		C=	***1₋₁***	2	2	3	3	3	4	5	5
	1	2	3	4	5	6			0	1	2	3	4	5	6	7	8
B=	0	0	2	6	6	8		C=	0	2	2	3	3	3	***4₋₁***	5	5
	1	2	3	4	5	6			0	1	2	3	4	5	6	7	8
							final	C=	0	2	2	3	3	3	3	5	5
									0	1	2	3	4	5	6	7	8

$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$

$h(k,i) = (k^2 + i + 2i^2) \bmod 5$

Open addressing using quadratic probing

$h(3,0) = (3^2 + 0 + 0) \bmod 5 = 4$

$h(4,0) = (4^2 + 0 + 0) \bmod 5 = 1$

$h(2,0) = (2^2 + 0 + 0) \bmod 5 = 4$ (collision)

$h(2,1) = (2^2 + 1 + 2) \bmod 5 = 2$

$h(5,0) = (5^2 + 0 + 0) \bmod 5 = 0$

$h(1,0) = (1^2 + 0 + 0) \bmod 5 = 1$ (collision)

$h(1,1) = (1^2 + 1 + 2) \bmod 5 = 4$ (collision)

$h(1,2) = (1^2 + 2 + 8) \bmod 5 = 1$ (collision)

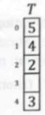
$h(1,3) = (1^2 + 3 + 18) \bmod 5 = 2$ (collision)

$h(1,4) = (1^2 + 4 + 32) \bmod 5 = 2$ (collision)

$h(1,0) = (1^2 + 0 + 0) \bmod 5 = 1$ (collision)

⋮

No spot for $k = 1$



(4) What different values can the hash function $h(k) = k^2 \bmod m$ produce when $m = 11$? Carefully justify your answer in detail.

0 mod 11 = 0	121 mod 11 = 0
1 mod 11 = 1	144 mod 11 = 1
4 mod 11 = 4	169 mod 11 = 4
9 mod 11 = 9	196 mod 11 = 9
16 mod 11 = 5	225 mod 11 = 5
25 mod 11 = 3	256 mod 11 = 3
36 mod 11 = 3	289 mod 11 = 3
49 mod 11 = 5	324 mod 11 = 5
64 mod 11 = 9	361 mod 11 = 9
81 mod 11 = 4	400 mod 11 = 4
100 mod 11 = 1	441 mod 11 = 1

The pattern will continue repeating,

$\therefore \Rightarrow \langle 0,1,3,4,5,9 \rangle$

Building a heap:

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

Running time of Quicksort

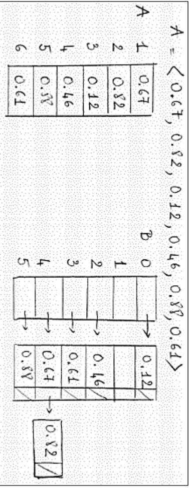
$T(n) = \Theta(n) + T(q-1) + T(n-q)$

$= T(\alpha n) + T((1-\alpha)n) + \Theta(n)$

Mergesort $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

$$T(n) = 3T(n/4) + cn^2.$$

$A = \langle 0.67, 0.82, 0.12, 0.46, 0.88, 0.61 \rangle$



floor (0.67 × 6) = floor (4.02) = 4

floor (0.82 × 6) = floor (4.92) = 4

floor (0.12 × 6) = floor (0.72) = 0

floor (0.46 × 6) = floor (2.76) = 2

floor (0.88 × 6) = floor (5.28) = 5

floor (0.61 × 6) = flar (3.66) = 3

Answer : 0.12, 0.46, 0.61, 0.67, 0.82, 0.88

The operation HEAP-DELETE (A, i) deletes the item in node i from heap A. Give an implementation of HEAP-DELETE that runs in O(*log* n) time for an *n*-element max-heap.

(1) Pseudocode

```
                                                        Cost   # of executions
HEAP-DELETE (A, A.heap-size, i)
(1)   A[i] = A[A.heap-size]                              C1     1
          /* overwrites the value of A[i] by the value of the smallest leaf */
(2)   A.heap-size -= 1                                   C2     1
(3)   while (i > 1 and parent of i < A[i])               C3     1
(4)       swap (A[parent(i)]) with A[i]                  C4     1
(5)       i = parent(i)                                  C5     1
(6)   MAX-HEAPIFY(A, i)                                  C6     1
```

(2) provide an upper bound of your procedure and give an explanation

Explanation: as of A[i] has been removed and replaced by the smallest leaf, A[i] must be iteratively compared with its parent then swap if the parent is ≤ A[i] to ensure A[parent(i)] ≥ A[i] (line 3, 4, 5).

In case of A[i] is small, thus it recursively swaps with its child and traverses the longest path until it becomes a leaf.

$T(n) = C1*1 + C2*1 + C3*\sum_2^h 1 + C4*\sum_2^h 1 + C5*\sum_2^h 1 + C6*1$

$= C1 + C2 + (C3 + C4 + C5)*(h-2+1) + C6$

$= C*(h-1) + D$ (C and D are positive constants)

$= O(h) + D$ (provided h = *lg* n)

$= O(lg\ n)$ = > proved

n : number of elements

m : number of slots

$h(k) = h'(k) \bmod m$ **Using chaining**

$h(k) = k^2 \bmod 5$

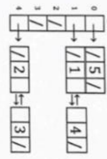
$h(3) = (3^2 \bmod 5) = 4$

$h(4) = (4^2 \bmod 5) = 1$

$h(2) = (2^2 \bmod 5) = 4$

$h(5) = (5^2 \bmod 5) = 0$

$h(1) = (1^2 \bmod 5) = 1$



$\alpha = \frac{n}{m} = \frac{5}{5} = 1$

(2) Using $h(k)$ as the primary hash function, illustrate the result of inserting these keys using open addressing with linear probing.

$h(k,i) = (h'(k) + i) \bmod m$

$h(k,i) = (k^2 + i) \bmod 5$

$h(3,0) = [(3^2 + 0) \bmod 5] = 4$

$h(4,0) = [(4^2 + 0) \bmod 5] = 1$

$h(2,0) = [(2^2 + 0) \bmod 5] = 4$ (collision)

$h(2,1) = [(2^2 + 1) \bmod 5] = 0$

$h(5,0) = [(5^2 + 0) \bmod 5] = 0$ (collision)

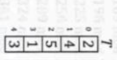
$h(5,1) = [(5^2 + 1) \bmod 5] = 1$ (collision)

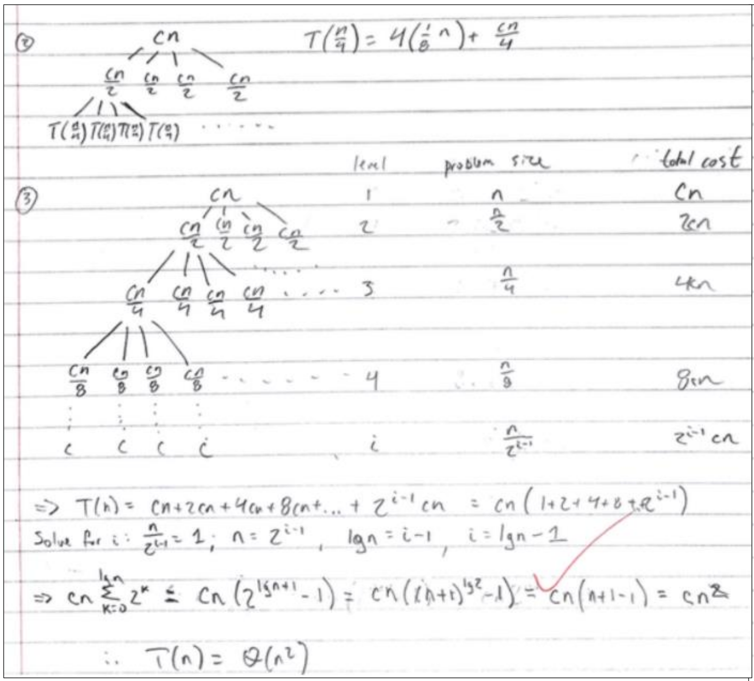
$h(5,2) = [(5^2 + 2) \bmod 5] = 2$

$h(1,0) = [(1^2 + 0) \bmod 5] = 1$ (collision)

$h(1,1) = [(1^2 + 1) \bmod 5] = 2$ (collision)

$h(1,2) = [(1^2 + 2) \bmod 5] = 3$





$T(\frac{n}{4}) = 4(\frac{1}{8}n) + \frac{cn}{4}$

③
level
problem size
total cost
cn
$\frac{cn}{2}$ $\frac{cn}{2}$ $\frac{cn}{2}$ $\frac{cn}{2}$
$\frac{cn}{4}$ $\frac{cn}{4}$ $\frac{cn}{4}$ $\frac{cn}{4}$ . . . .
$\frac{cn}{8}$ $\frac{cn}{8}$ $\frac{cn}{8}$ $\frac{cn}{8}$ . . . . . .
c c c c

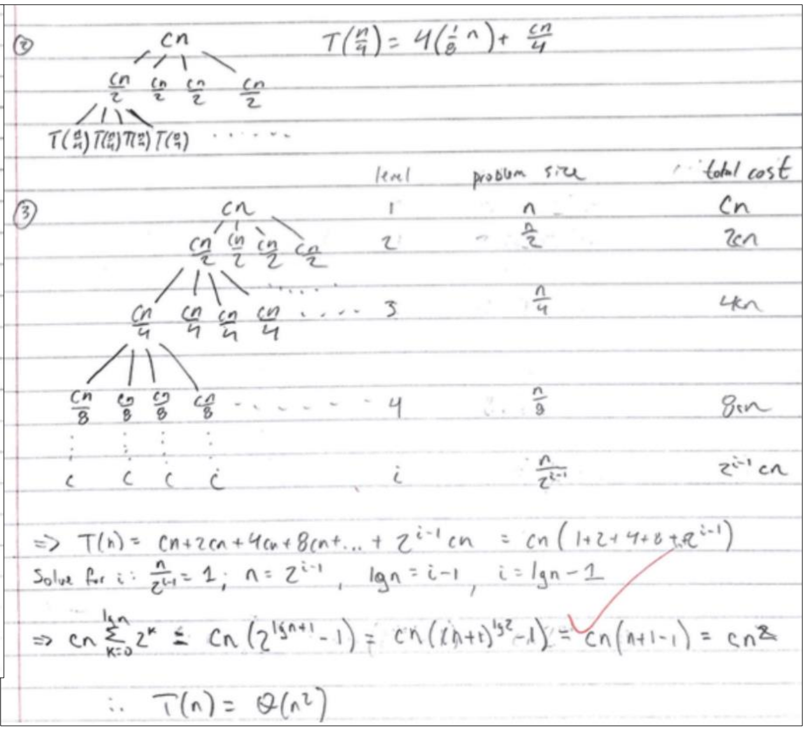
level	problem size	total cost
1	n	cn
2	$\frac{n}{2}$	$2cn$
3	$\frac{n}{4}$	$4cn$
4	$\frac{n}{8}$	$8cn$
i	$\frac{n}{2^{i-1}}$	$2^{i-1}cn$

$\Rightarrow T(n) = cn + 2cn + 4cn + 8cn + \ldots + 2^{i-1}cn = cn(1+2+4+8+2^{i-1})$

Solve for i: $\frac{n}{2^{i-1}} = 1$; $n = 2^{i-1}$, $\lg n = i-1$, $i = \lg n - 1$

$\Rightarrow cn\sum_{k=0}^{\lg n} 2^k \leq cn(2^{\lg n+1}-1) = cn((n+1)^{\lg 2}-1) = cn(n+1-1) = cn^2$

$\therefore T(n) = \Theta(n^2)$


②
cn
$\frac{cn}{2}$ $\frac{cn}{2}$ $\frac{cn}{2}$ $\frac{cn}{2}$
$T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ . . . . . .

$T(\frac{n}{4}) = 4(\frac{1}{8}n) + \frac{cn}{4}$

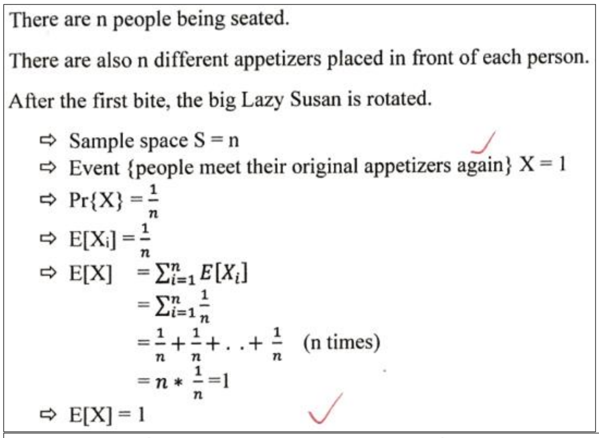
③
level problem size total cost
1 n cn
2 $\frac{n}{2}$ $2cn$
3 $\frac{n}{4}$ $4cn$
4 $\frac{n}{8}$ $8cn$
i $\frac{n}{2^{i-1}}$ $2^{i-1}cn$

$\Rightarrow T(n) = cn + 2cn + 4cn + 8cn + \ldots + 2^{i-1}cn = cn(1+2+4+8+2^{i-1})$

Solve for i: $\frac{n}{2^{i-1}} = 1$; $n = 2^{i-1}$, $\lg n = i-1$, $i = \lg n - 1$

$\Rightarrow cn\sum_{k=0}^{\lg n} 2^k \leq cn(2^{\lg n+1}-1) = cn((n+1)^{\lg 2}-1) = cn(n+1-1) = cn^2$

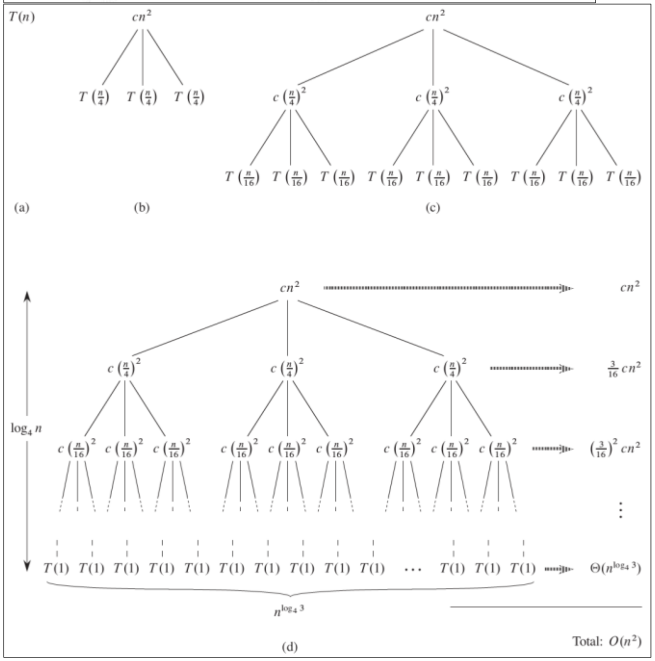
$\therefore T(n) = \Theta(n^2)$

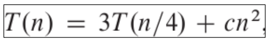

There are n people being seated.

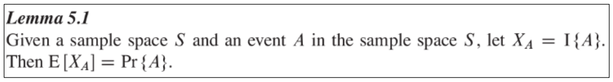
There are also n different appetizers placed in front of each person.

After the first bite, the big Lazy Susan is rotated.

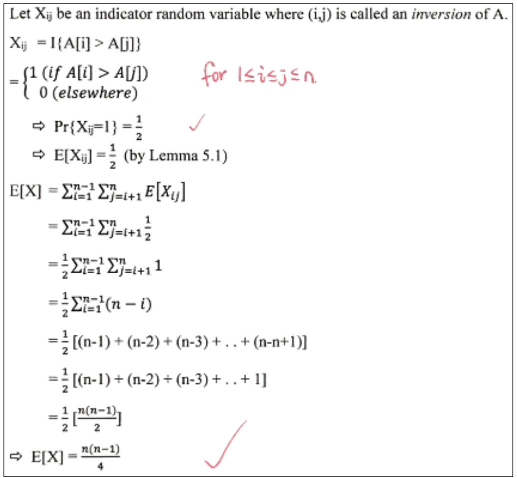
- ⇨ Sample space S = n
- ⇨ Event {people meet their original appetizers again} X = 1
- ⇨ $\Pr\{X\} = \frac{1}{n}$
- ⇨ $E[X_i] = \frac{1}{n}$
- ⇨ $E[X] = \sum_{i=1}^{n} E[X_i]$
  $= \sum_{i=1}^{n} \frac{1}{n}$
  $= \frac{1}{n} + \frac{1}{n} + \ldots + \frac{1}{n}$ (n times)
  $= n * \frac{1}{n} = 1$
- ⇨ E[X] = 1


$T(n)$
cn^2
$T(\frac{n}{4})$ $T(\frac{n}{4})$ $T(\frac{n}{4})$
cn^2
$c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$
$T(\frac{n}{16})$ $T(\frac{n}{16})$ $T(\frac{n}{16})$ $T(\frac{n}{16})$ $T(\frac{n}{16})$ $T(\frac{n}{16})$ $T(\frac{n}{16})$ $T(\frac{n}{16})$ $T(\frac{n}{16})$
(a)
(b)
(c)
cn^2
$c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$ $c(\frac{n}{4})^2$
$c(\frac{n}{16})^2$ $c(\frac{n}{16})^2$ $c(\frac{n}{16})^2$ $c(\frac{n}{16})^2$ $c(\frac{n}{16})^2$ $c(\frac{n}{16})^2$ $c(\frac{n}{16})^2$ $c(\frac{n}{16})^2$ $c(\frac{n}{16})^2$
$\log_4 n$
cn^2
$\frac{3}{16}cn^2$
$(\frac{3}{16})^2 cn^2$
⋮
$T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ … $T(1)$ $T(1)$ $T(1)$
$\Theta(n^{\log_4 3})$
$n^{\log_4 3}$
(d)
Total: $O(n^2)$


$$T(n) = 3T(n/4) + cn^2,$$


Lemma 5.1

Given a sample space S and an event A in the sample space S, let $X_A = \mathrm{I}\{A\}$. Then $\mathrm{E}[X_A] = \Pr\{A\}$.


Let X_{ij} be an indicator random variable where (i,j) is called an *inversion* of A.

$X_{ij} = I\{A[i] > A[j]\}$

$= \begin{cases} 1\ (if\ A[i] > A[j]) \\ 0\ (elsewhere) \end{cases}$ for $1 \leq i \leq j \leq n$

- ⇨ $\Pr\{X_{ij}=1\} = \frac{1}{2}$ ✓
- ⇨ $E[X_{ij}] = \frac{1}{2}$ (by Lemma 5.1)

$E[X] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$

$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{1}{2}$

$= \frac{1}{2}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1$

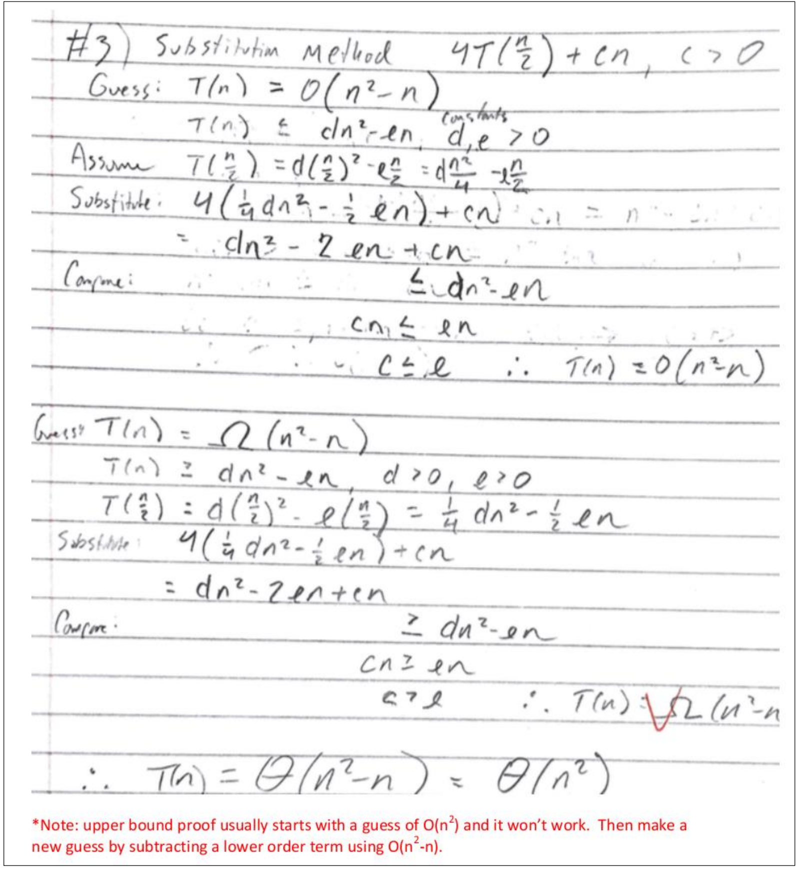
$= \frac{1}{2}\sum_{i=1}^{n-1}(n-i)$

$= \frac{1}{2}[(n-1) + (n-2) + (n-3) + \ldots + (n-n+1)]$

$= \frac{1}{2}[(n-1) + (n-2) + (n-3) + \ldots + 1]$

$= \frac{1}{2}[\frac{n(n-1)}{2}]$

⇨ $E[X] = \frac{n(n-1)}{4}$ ✓


#3) Substitution Method $4T(\frac{n}{2}) + cn$, $c > 0$

Guess: $T(n) = O(n^2 - n)$

$T(n) \leq dn^2 - en$, constants $d, e > 0$

Assume $T(\frac{n}{2}) = d(\frac{n}{2})^2 - e\frac{n}{2} = d\frac{n^2}{4} - e\frac{n}{2}$

Substitute: $4(\frac{1}{4}dn^2 - \frac{1}{2}en) + cn$

$= dn^2 - 2en + cn$

Compare: $\leq dn^2 - en$

$cn \leq en$

$c \leq e$ $\therefore T(n) = O(n^2 - n)$

Guess: $T(n) = \Omega(n^2 - n)$

$T(n) \geq dn^2 - en$, $d > 0$, $e > 0$

$T(\frac{n}{2}) = d(\frac{n}{2})^2 - e(\frac{n}{2}) = \frac{1}{4}dn^2 - \frac{1}{2}en$

Substitute: $4(\frac{1}{4}dn^2 - \frac{1}{2}en) + cn$

$= dn^2 - 2en + cn$

Compare: $\geq dn^2 - en$

$cn \geq en$

$c \geq e$ $\therefore T(n) = \Omega(n^2 - n$

$\therefore T(n) = \Theta(n^2 - n) = \Theta(n^2)$

*Note: upper bound proof usually starts with a guess of O(n^2) and it won't work. Then make a new guess by subtracting a lower order term using O(n^2-n).