5.2-1

In HIRE-ASSISTANT, assuming that the candidates are presented in a random order, what is the probability that you hire exactly one time? What is the probability that you hire exactly n times?

P(hire exactly time) : P(the first is the best)

$= \frac{(n+1)!}{n!} = \frac{1}{n}$ #

P(hire exactly n times) = P(they are in increasing order)

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

5.2-6

Let $A[1:n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an ***inversion*** of A. (See Problem 2-4 on page 47 for more on inversions.) Suppose that the elements of A form a uniform random permutation of $\langle 1, 2, \ldots, n \rangle$. Use indicator random variables to compute the expected number of inversions.

$X_{ij} = \{ A[i] > A[j] \}$ for $1 \le i \le j \le n$

$P(X_{ij} = 1) = \frac{1}{2} \Rightarrow E[X_{ij}] = \frac{1}{2}$

X = total pairs of inversion

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

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

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

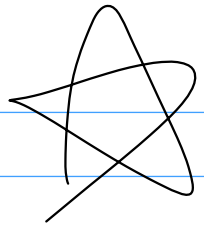
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2}$$

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

$$= \frac{n(n-1)}{4} \quad \#$$

5.4-2

How many people must there be in a room before the probability that two people have the same birthday is at least 0.99? For that many people, what is the expected number of pairs of people who have the same birthday?



P(none of n people have the same birthday)

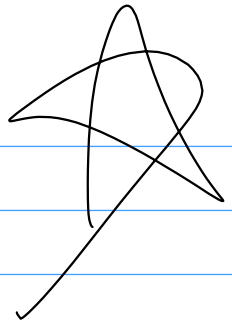
$= \left(1 - \frac{1}{365}\right)^n = \left(\frac{364}{365}\right)^n \geq 0.99$

$\Rightarrow n \log \frac{364}{365} \geq \log 0.99$

$\Rightarrow n \geq \frac{\log 0.99}{\log 364 - \log 365} \approx 3.7 \Rightarrow n = 4$

5.4-3

You toss balls into b bins until some bin contains two balls. Each toss is independent, and each ball is equally likely to end up in any bin. What is the expected number of ball tosses?



X_i := the bin i is empty after all balls are tossed

6.1-4

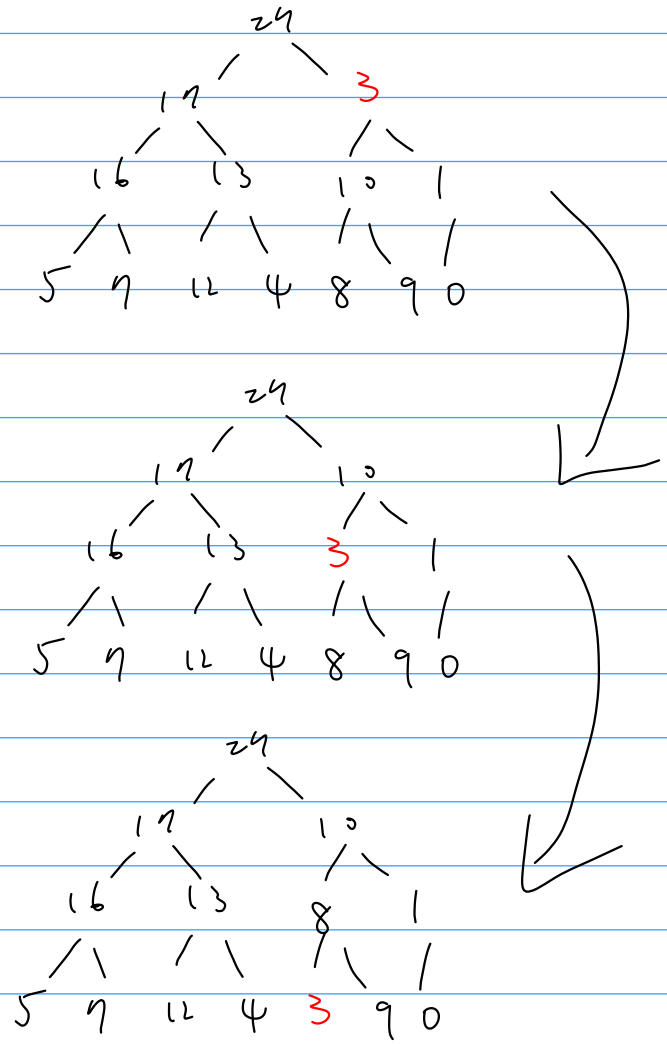
Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

∵ parent > children

∴ the smallest element will be one of leaf node.

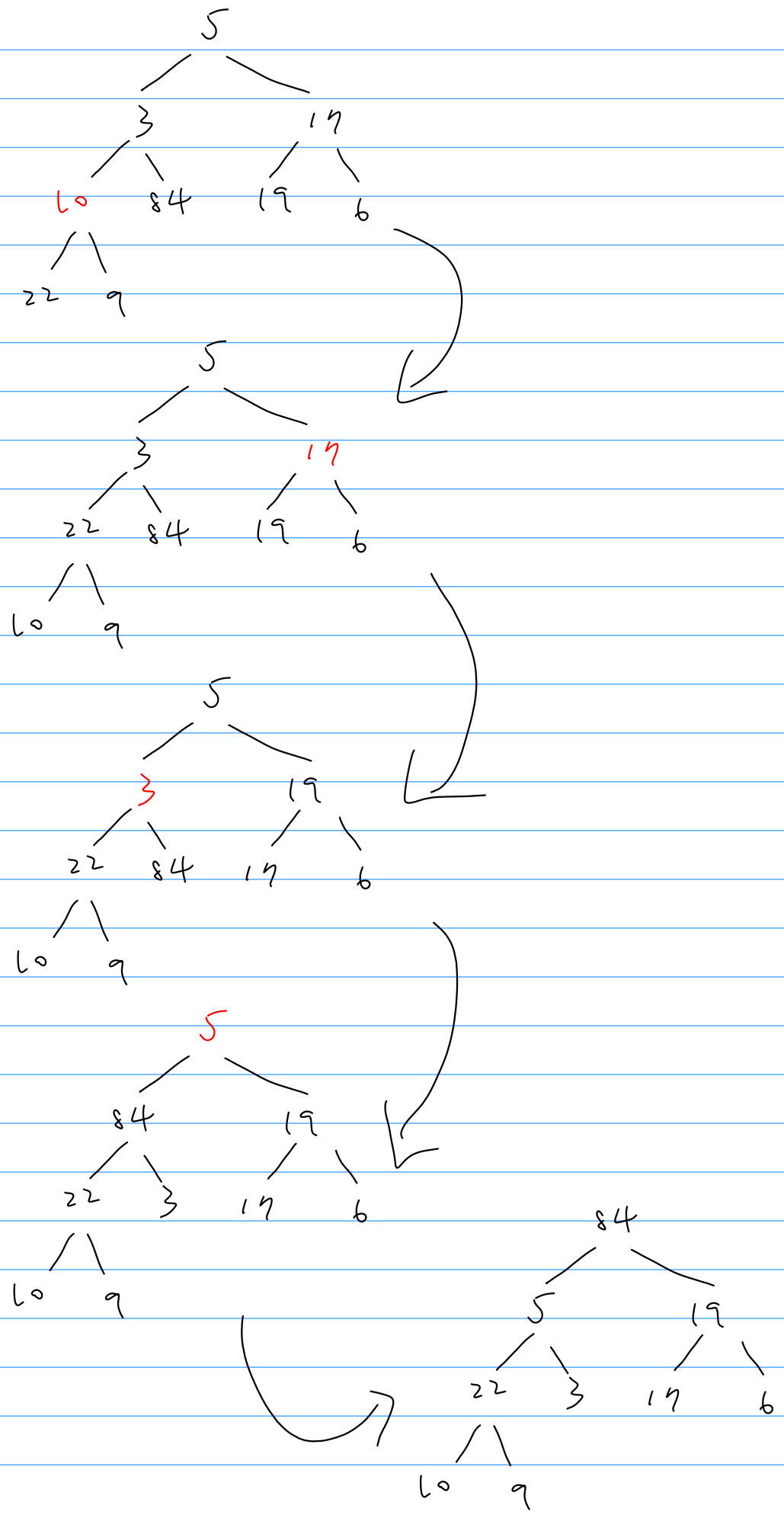
6.2-1

Using Figure 6.2 as a model, illustrate the operation of MAX-HEAPIFY$(A, 3)$ on the array $A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.



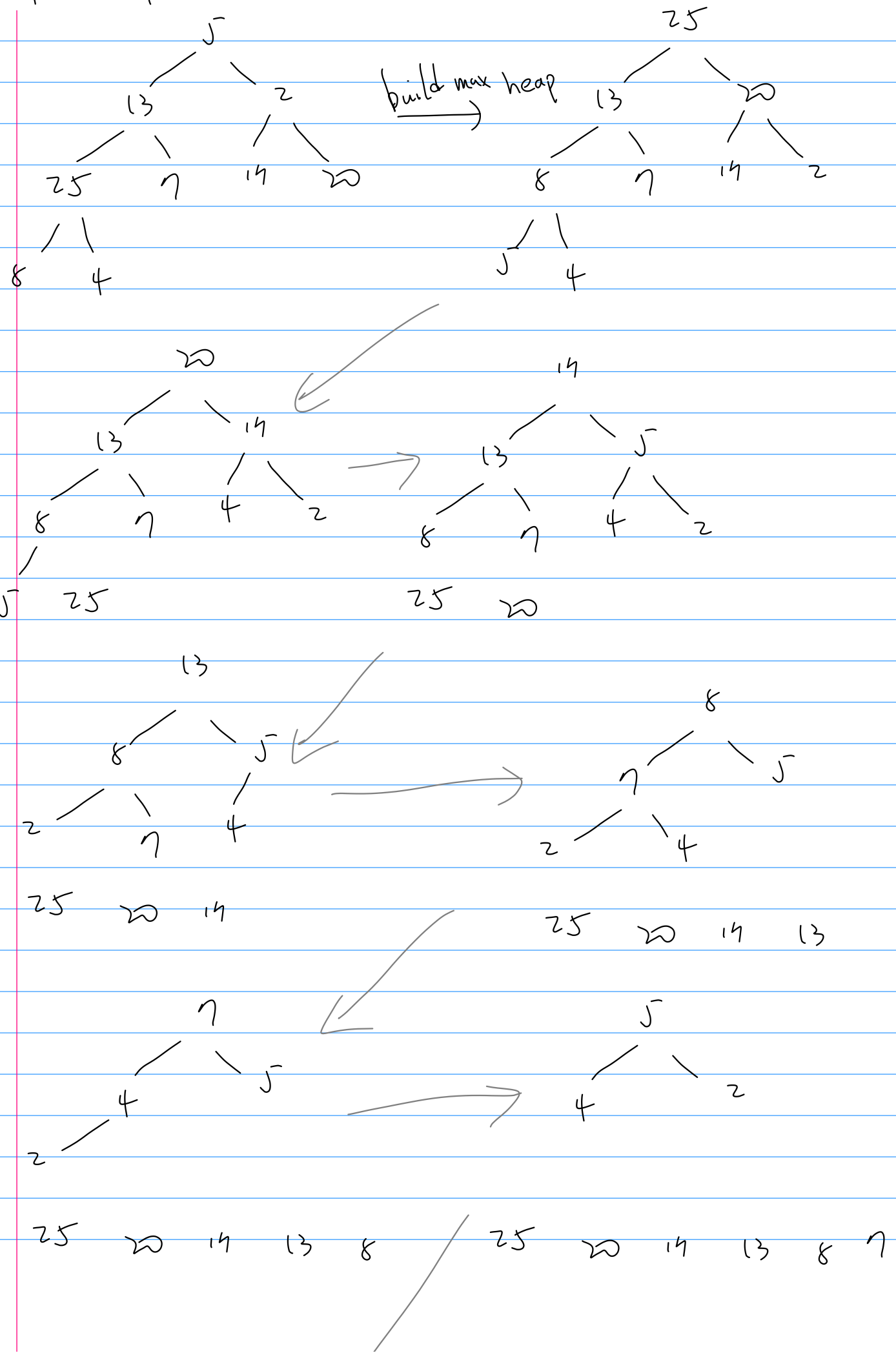
6.3-1

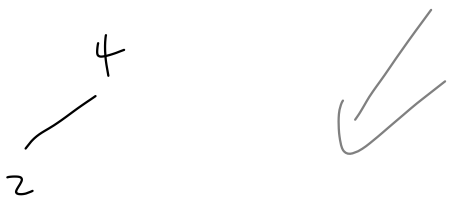
Using Figure 6.3 as a model, illustrate the operation of BUILD-MAX-HEAP on the array $A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.



6.4-1

Using Figure 6.4 as a model, illustrate the operation of HEAPSORT on the array $A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.





25 20 14 13 8 7 5

25 20 14 13 8 7 5 4 2

6.4-4

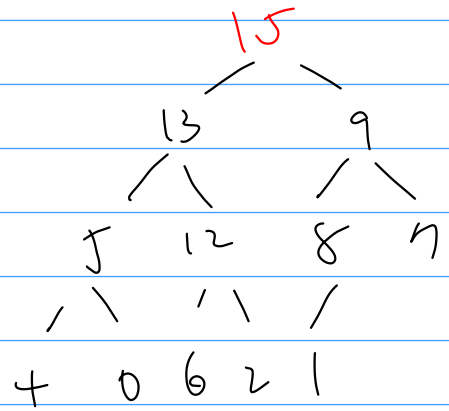
Show that the worst-case running time of HEAPSORT is $\Omega(n \lg n)$.

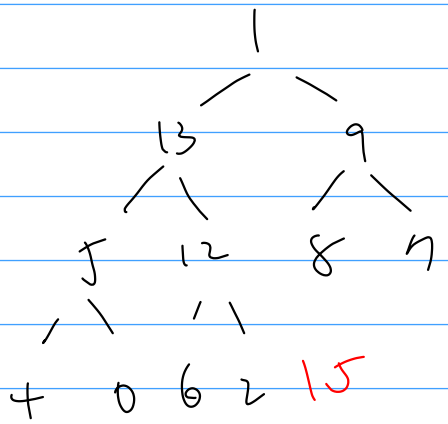
$$\sum_{i=1}^{h} 2^i \log(2^i) = \sum_{i=1}^{\lfloor \lg n \rfloor} i \, 2^i$$

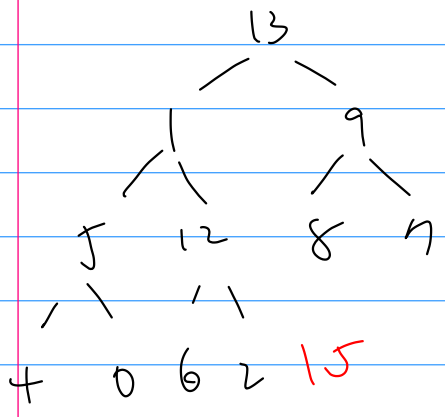
$$= 2 + (\lfloor \lg n \rfloor - 1) 2^{\lfloor \lg n \rfloor} = \Omega(n \lg n) \quad \#$$

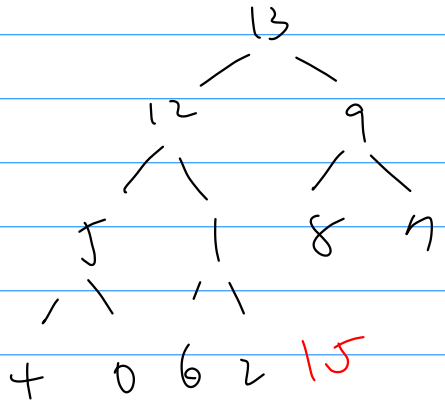
6.5-1

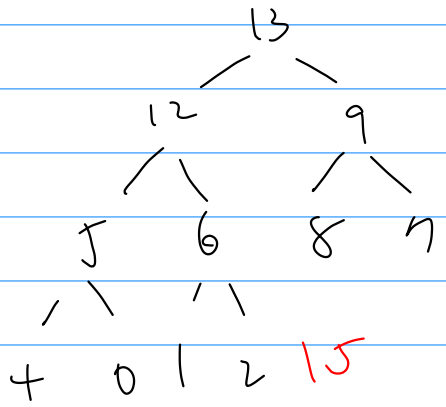
Suppose that the objects in a max-priority queue are just keys. Illustrate the operation of MAX-HEAP-EXTRACT-MAX on the heap $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.











6.5-4

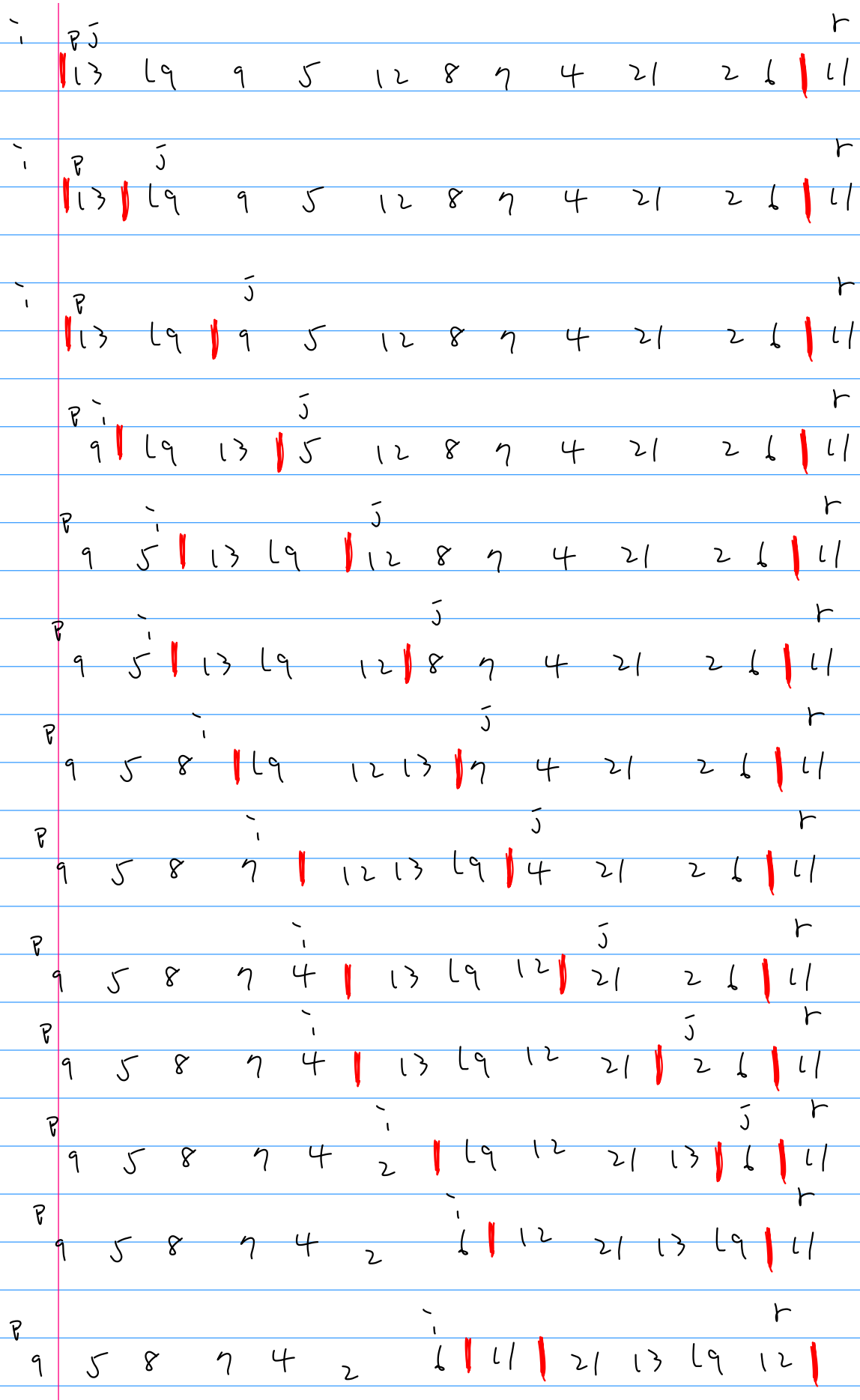
Write pseudocode for the procedure MAX-HEAP-DECREASE-KEY(A, x, k) in a max-heap. What is the running time of your procedure?

```
max-heap-decrease-key (A, x, k) {
    if k > x.key {
        error "new key is large the current key"
    }
    x.key = k
    find the index i in A where the object x occurs
    while i > 1 and
      ( A[LEFT(i)].key > A[i].key or A[RIGHT(i)].key > A[i].key){
        m = max (i, LEFT(i), RIGHT(i))
        exchange A[i], A[m]
        i = max
    }
}
```

$\Rightarrow \Omega(h) = \Omega(\lg n)$

7.1-1

Using Figure 7.1 as a model, illustrate the operation of PARTITION on the array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$.



7.1-3

Give a brief argument that the running time of PARTITION on a subarray of size n is $\Theta(n)$.

Partition only has a loop j run from head to tail of subarray, so it is $\Theta(n)$

7.2-2

What is the running time of QUICKSORT when all elements of array A have the same value?

Partition = $\Theta(n)$

it need to be run n times, so quicksort = $\Theta(n^2)$ #

7.2-4

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. People usually write checks in order by check number, and merchants usually cash them with reasonable dispatch. The problem of converting time-of-transaction ordering to check-number ordering is therefore the problem of sorting almost-sorted input. Explain persuasively why the procedure INSERTION-SORT might tend to beat the procedure QUICKSORT on this problem.

7.3-2

When RANDOMIZED-QUICKSORT runs, how many calls are made to the random-number generator RANDOM in the worst case? How about in the best case? Give your answer in terms of Θ-notation.

Random : Randomize-Partition $= \Theta(n)$

7.4-2

Show that quicksort's best-case running time is $\Omega(n \lg n)$.

$$T(n) = \min_{1 \le q \le n-1} \left(T(q) + T(n-1-q) \right) + \Theta(n)$$

suppose that $T(n) \ge c(n \lg n + 2n)$

$$\Rightarrow T(n) \ge \min \Big[cq \lg q + 2cq + c(n-1-q)\lg(n-1-q) + 2c(n-1-q) \Big] + \Theta(n)$$

$q = \frac{n}{2}$ ⟹ $\frac{cn}{2} \lg \frac{n}{2} + cn + c\left(\frac{n}{2}-1\right)\lg\left(\frac{n}{2}-1\right) + cn - 2c + \Theta(n)$

$\lg n - \lg 2 = \lg n - 1$

$= \frac{cn}{2} \lg \frac{n}{2} + c\left(\frac{n}{2}-1\right)\lg\left(\frac{n}{2}-1\right) + 2cn - 2c + \Theta(n)$

$\ge \frac{cn}{2} \lg n - \frac{cn}{2} + c\left(\frac{n}{2}-1\right)(\lg n - 2) + 2cn - 2c + \Theta(n)$

$\lg\left(\frac{n}{2}-1\right) \ge \lg n - 2$

$= \frac{cn}{2}\lg n - \frac{cn}{2} + \left(\frac{cn}{2}\lg n - cn - c\lg n + 2c\right) + 2cn - 2c + \Theta(1)$

$\lg\left(\frac{n-2}{2}\right)$

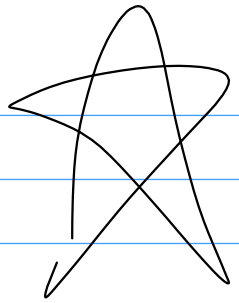
$\lg(n-2) - \lg 2 \ge \lg n - \lg 2 - \lg 2$

$= cn\lg n + \frac{cn}{2} + c\lg n + \Theta(1)$

$= \Omega(n \lg n)$ #

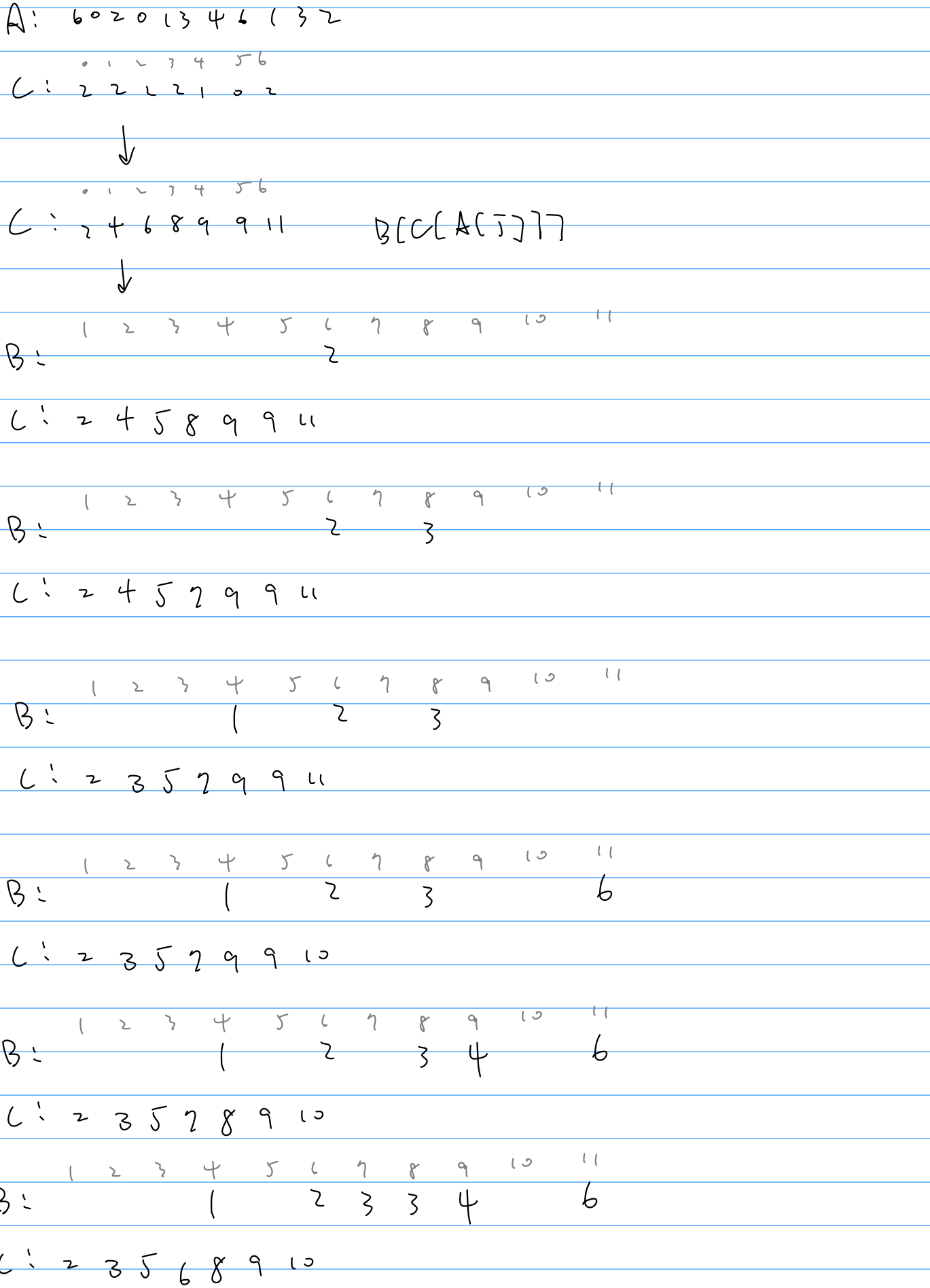
8.1-3

Show that there is no comparison sort whose running time is linear for at least half of the $n!$ inputs of length n. What about a fraction of $1/n$ of the inputs of length n? What about a fraction $1/2^n$?



8.2-1

Using Figure 8.2 as a model, illustrate the operation of COUNTING-SORT on the array $A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$.



1 2 3 4 5 6 7 8 9 10 11

B: 1 1 2 3 3 4 6

C: 2 2 5 6 8 9 10

1 2 3 4 5 6 7 8 9 10 11

B: 0 1 1 2 3 3 4 6

C: 1 2 5 6 8 9 10

1 2 3 4 5 6 7 8 9 10 11

B: 0 1 1 2 2 3 3 4 6

C: 1 2 4 6 8 9 10

1 2 3 4 5 6 7 8 9 10 11

B: 0 0 1 1 2 2 3 3 4 6

C: 0 2 4 6 8 9 10

1 2 3 4 5 6 7 8 9 10 11

B: 0 0 1 1 2 2 3 3 4 6 6

C: 0 2 4 6 8 9 9

⇒ 0 0 1 1 2 2 3 3 4 6 6

8.3-2

Which of the following sorting algorithms are stable: insertion sort, merge sort, heapsort, and quicksort? Give a simple scheme that makes any comparison sort stable. How much additional time and space does your scheme entail?

stable: insertion sort and merge sort

scheme:

convert [] keys to [] (keys, index)

for example [1, 7, 6, 3, 1]

→ [(1, 1), (7, 2), (6, 3), (3, 4), (1, 5)]

define $(a, b) < (c, d) \Leftrightarrow a \leq c$ and $b < d$

it double the space, but running time doesn't change

8.4-2

Explain why the worst-case running time for bucket sort is $\Theta(n^2)$. What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time $O(n \lg n)$?

if all elements go to the same bucket

the time of placing element is $\Theta(n)$

there are n elements, so it is $\Theta(n^2)$ #

---

we can replace the linked-list to merge sort,

The time of placing element is $\Theta(\lg n)$

then total time of sorting is $\Theta(n \lg n)$

9.2-2

Write an iterative version of RANDOMIZED-SELECT.

```
Randomized-select(A, p, r, i) {
  while p < r {
    q = randomized-partition(A, p, r)
    k = q - p + 1
    if i == k {
      return A[q]
    }
    if i < k {
      r = q - 1
    } else {
      p = q
      i = i - k
    }
  }
  return A[p]
}
```

9.3-1

In the algorithm SELECT, the input elements are divided into groups of 5. Show that the algorithm works in linear time if the input elements are divided into groups of 7 instead of 5.

9.3-3

Show how to use SELECT as a subroutine to make quicksort run in $O(n \lg n)$ time in the worst case, assuming that all elements are distinct.