CX2101 Algorithm Design and Analysis

Tutorial 1 (Sorting)

Week 5: Q7-Q9

This Tutorial

- Heapsort

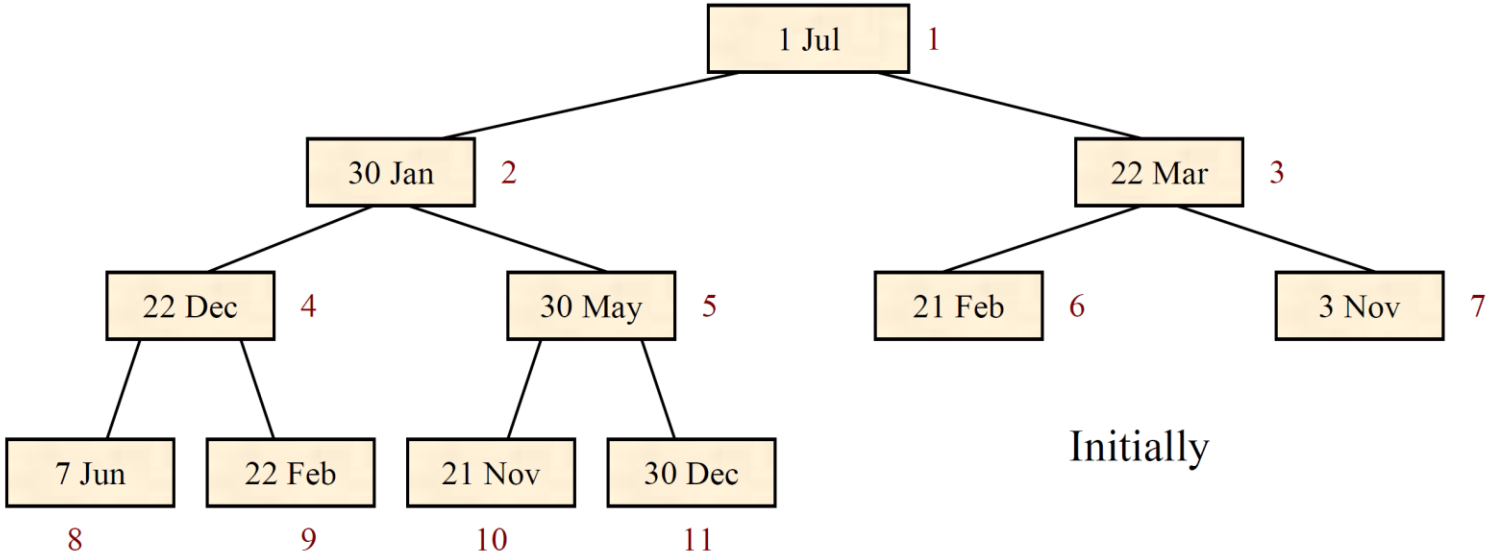
- Algorithm Design

Question 7

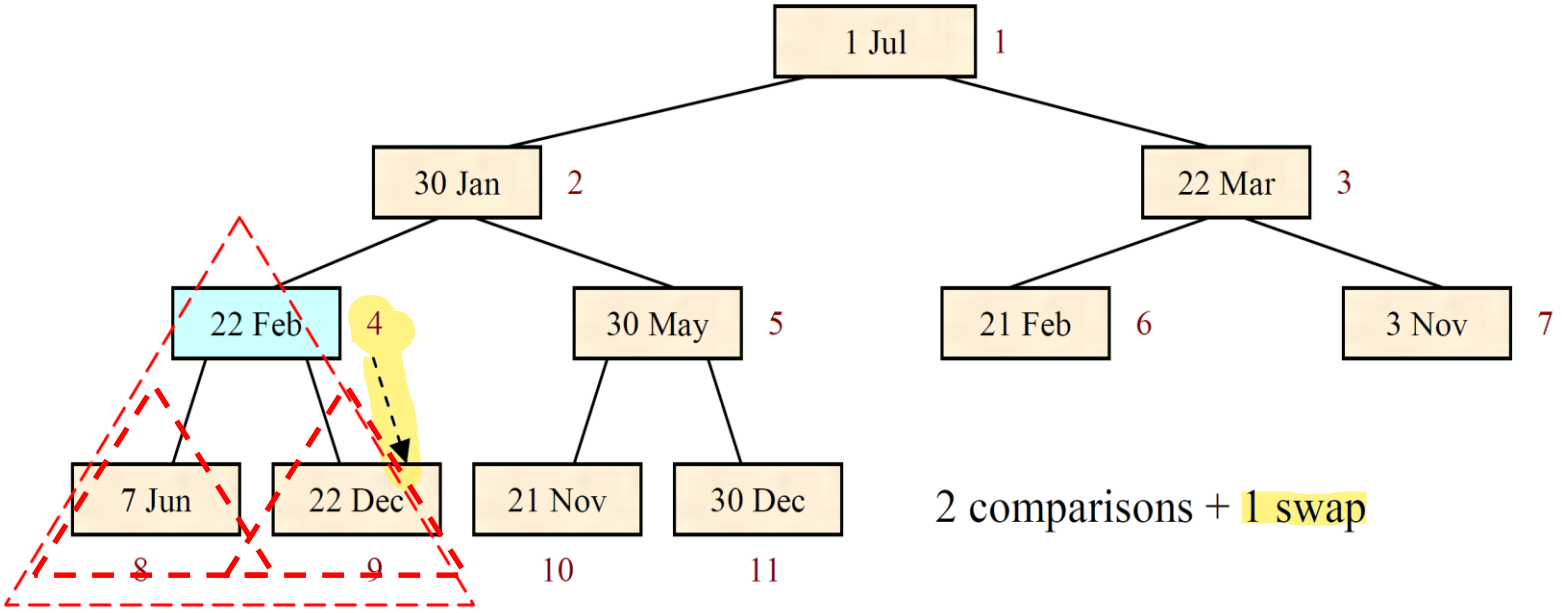
- Given an array with content (of type date): 1 Jul, 30 Jan, 22 Mar, 22 Dec, 30 May, 21 Feb, 3 Nov, 7 Jun, 22 Feb, 21 Nov, 30 Dec; all of the same year.
- Suppose an earlier date is considered bigger than a later date; for example, “30 Jan” is bigger than “30 Dec”.
- In order to sort these dates by Heapsort, let us construct a maximising heap. Show the content of the array after the heap construction phase.

1	2	3	4	5	6	7	8	9	10	11
1 Jul	30 Jan	22 Mar	22 Dec	30 May	21 Feb	3 Nov	7 Jun	22 Feb	21 Nov	30 Dec

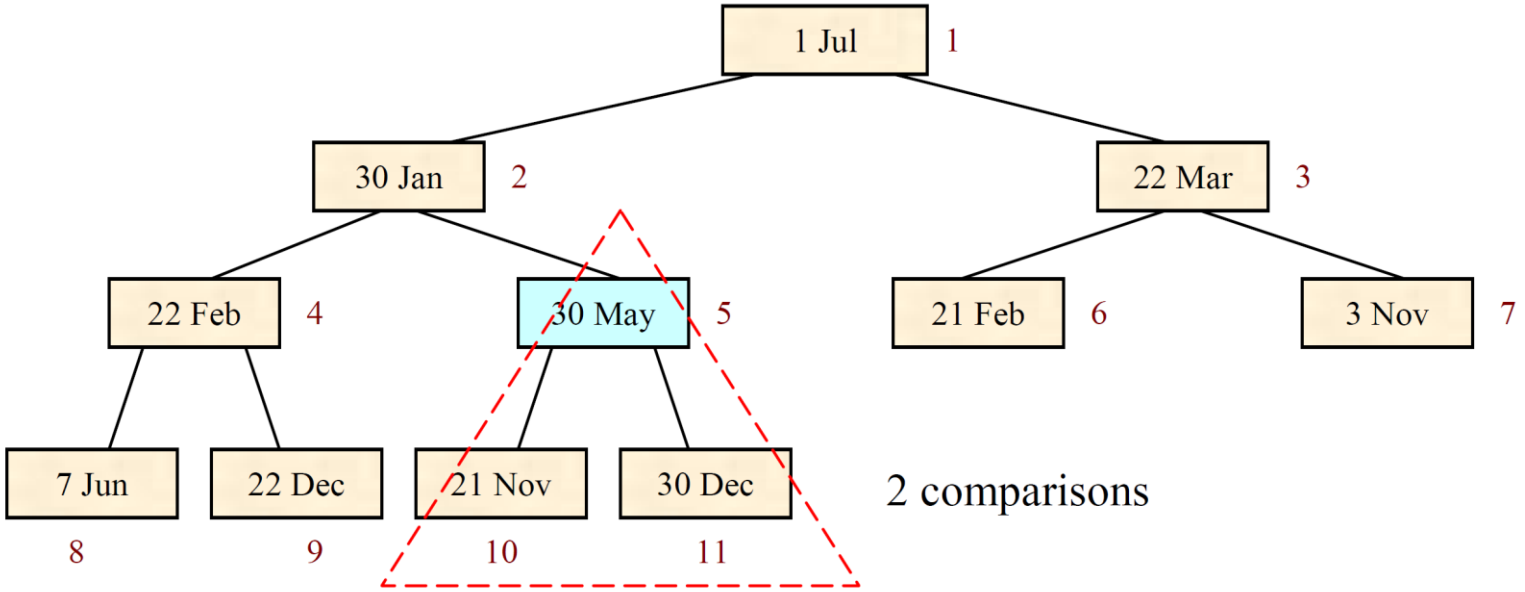
1. create heap based on array order
2. Compare parent with both childs and count swaps



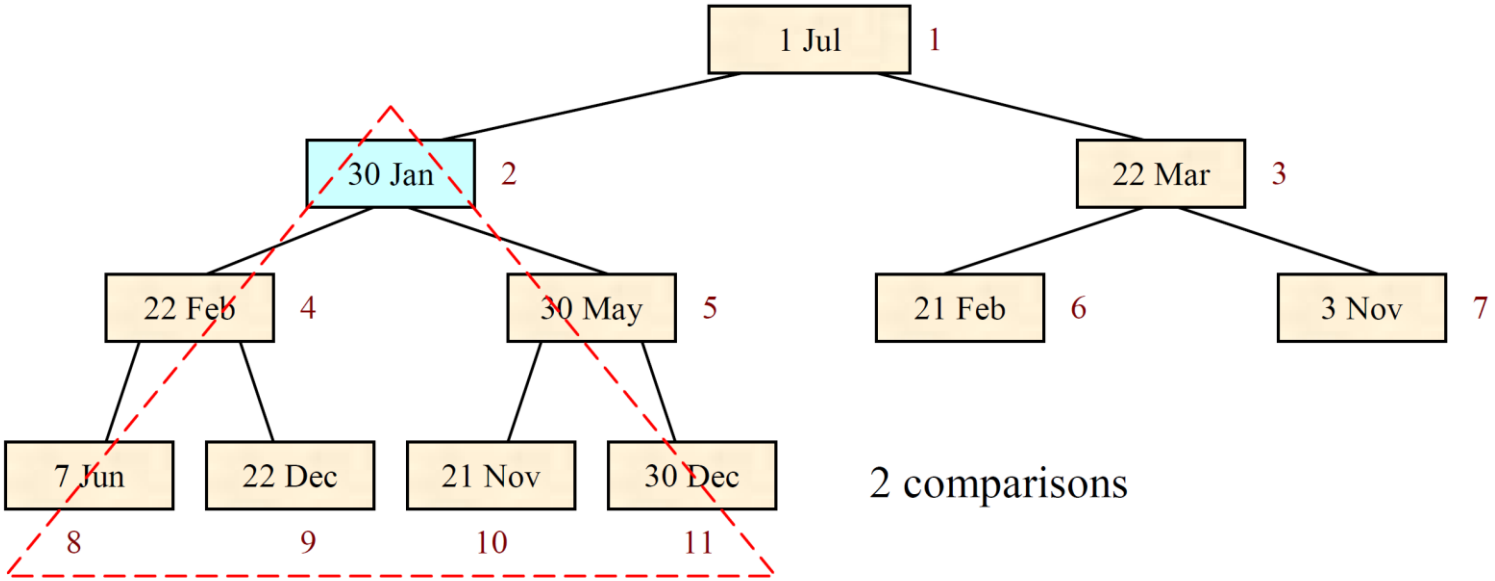
1	2	3	4	5	6	7	8	9	10	11
1 Jul	30 Jan	22 Mar	22 Dec	30 May	21 Feb	3 Nov	7 Jun	22 Feb	21 Nov	30 Dec



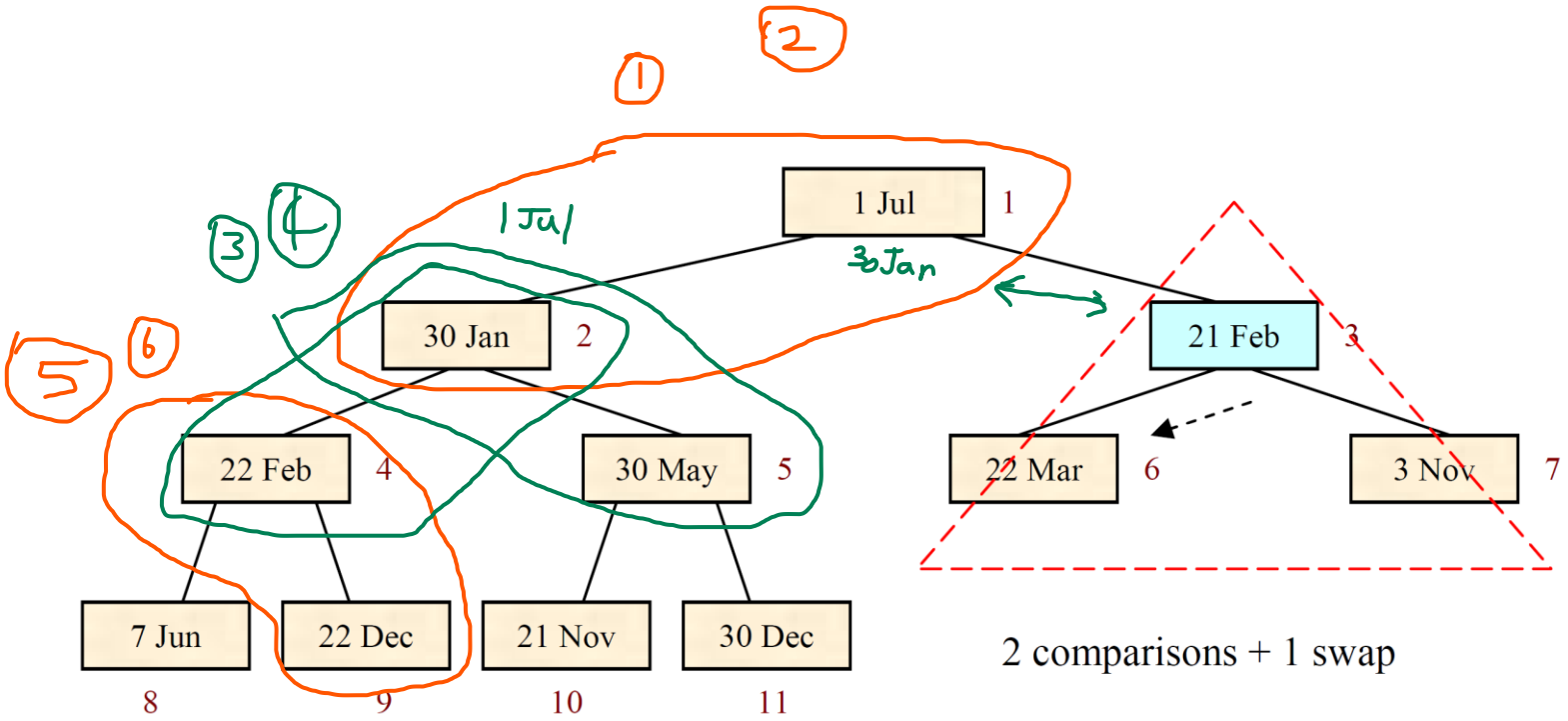
1	2	3	4	5	6	7	8	9	10	11
1 Jul	30 Jan	22 Mar	22 Feb	30 May	21 Feb	3 Nov	7 Jun	22 Dec	21 Nov	30 Dec



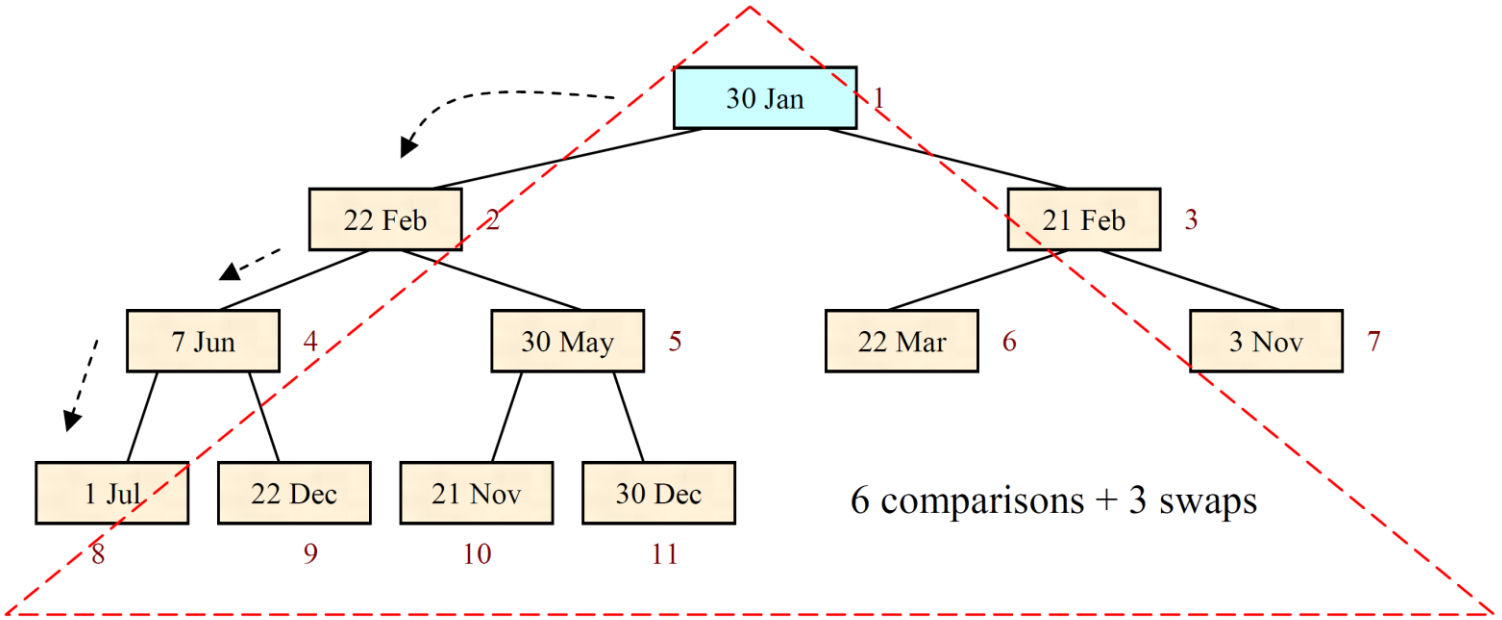
1	2	3	4	5	6	7	8	9	10	11
1 Jul	30 Jan	22 Mar	22 Feb	30 May	21 Feb	3 Nov	7 Jun	22 Dec	21 Nov	30 Dec



1	2	3	4	5	6	7	8	9	10	11
1 Jul	30 Jan	22 Mar	22 Feb	30 May	21 Feb	3 Nov	7 Jun	22 Dec	21 Nov	30 Dec



1	2	3	4	5	6	7	8	9	10	11
1 Jul	30 Jan	21 Feb	22 Feb	30 May	22 Mar	3 Nov	7 Jun	22 Dec	21 Nov	30 Dec

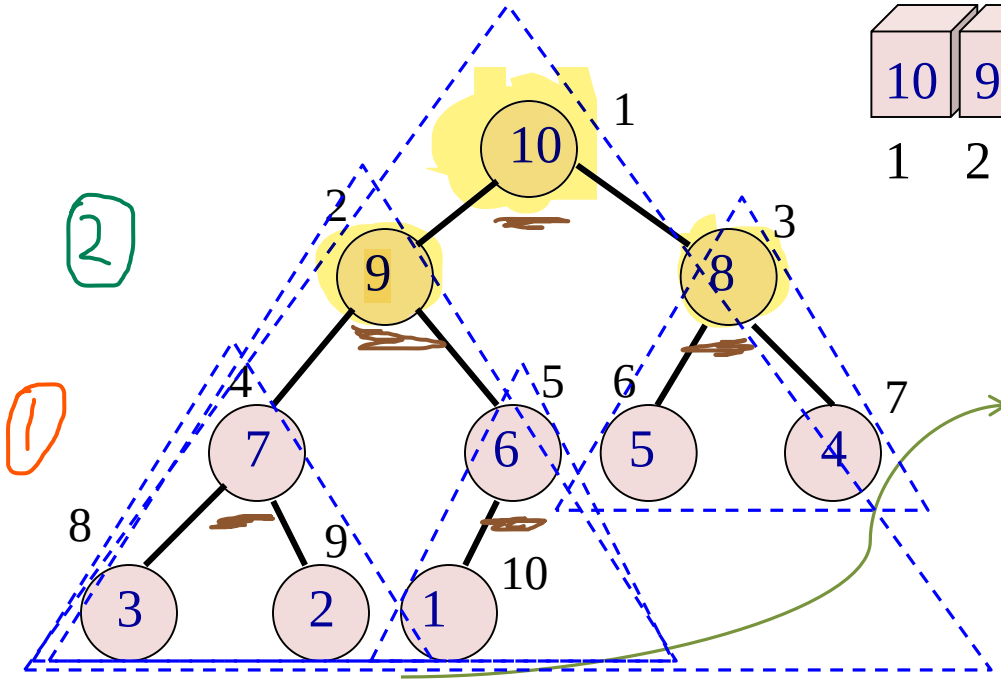


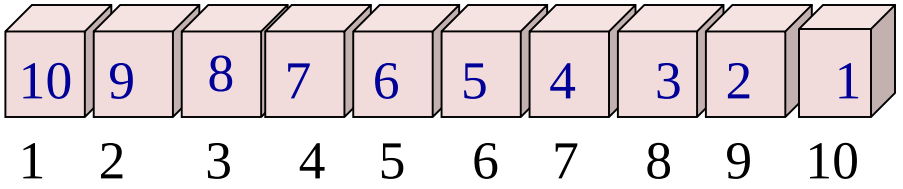
- **Ans**: Heapifying recursively performs **14** date comparisons and **5** swaps in the heap construction phase, and the content in the resulting array becomes:

1	2	3	4	5	6	7	8	9	10	11
30 Jan	22 Feb	21 Feb	7 Jun	30 May	22 Mar	3 Nov	1 Jul	22 Dec	21 Nov	30 Dec

Question 8

- An array of distinct keys in decreasing order is to be sorted (into increasing order) by HeapSort.
- (a) How many comparisons of keys are done in the heap construction phase (Algorithm constructHeap() of lecture notes) if there are 10 elements?
- **Ans**: 2 + 1 + 2 + 2 + 2 = 9.





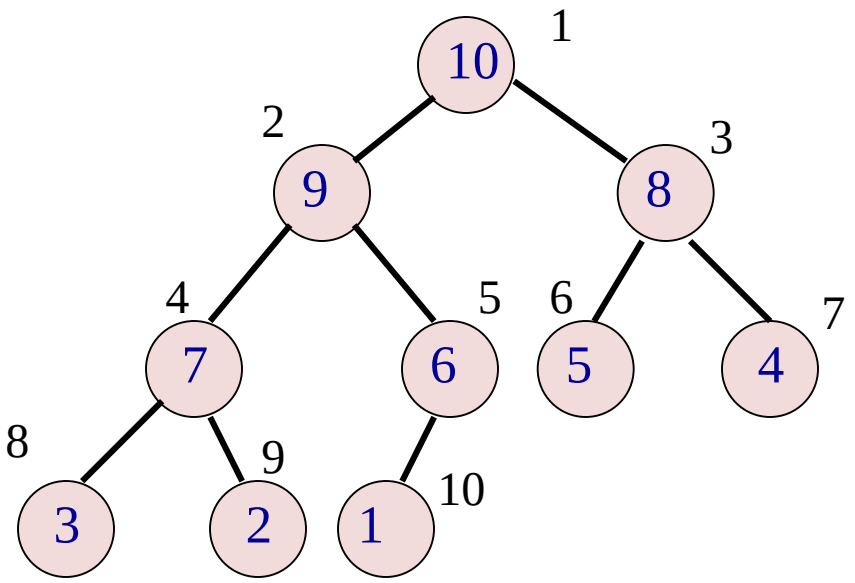
ID of nodes to be compared	No. comparisons
4, 8, 9	2
5, 10	1
2, 4, 5	2
3, 6, 7	2
1, 2, 3	2

Question 8

- (b) How many are done if there are n elements? Show how you derive your answer.
- **Ans**: Node of index i is an *internal* node (that is, not a *leaf*) if and only if $1 \le 2i \le n$, so $½ \le i \le n/2$. Thus,
  - If n is even, there are $n/2$ internal nodes
  - If n is odd, there are $(n – 1)/2$ internal nodes
- Since the keys are in decreasing order, FixHeap does only one or two key comparisons each time it is called; no key move
- If n is **even**, the last internal node has only one child, so
  - FixHeap does only one key comparison at that node
  - For each other internal node, two key comparisons are done
  - Total number of key comparisons is $2(n/2 – 1) + 1 = n – 1$
- If n is **odd**, each of the $(n – 1)/2$ internal nodes has two children, so
  - Two key comparisons are done by FixHeap for each internal node
  - Total number of key comparisons is $2 \times (n – 1)/2 = n – 1$
- In both cases, the total is $n – 1$ key comparisons

Question 8

- (c) Is an array in decreasing order a best case, worst case, or intermediate case for this algorithm? Justify your answer.
- **Ans**: It is the best case.
- Because there is no key move, so in the heapifying() algorithm, the fixHeap() function only compares the root and its two children, and no need to call fixHeap() recursively as in intermediate or worst case.

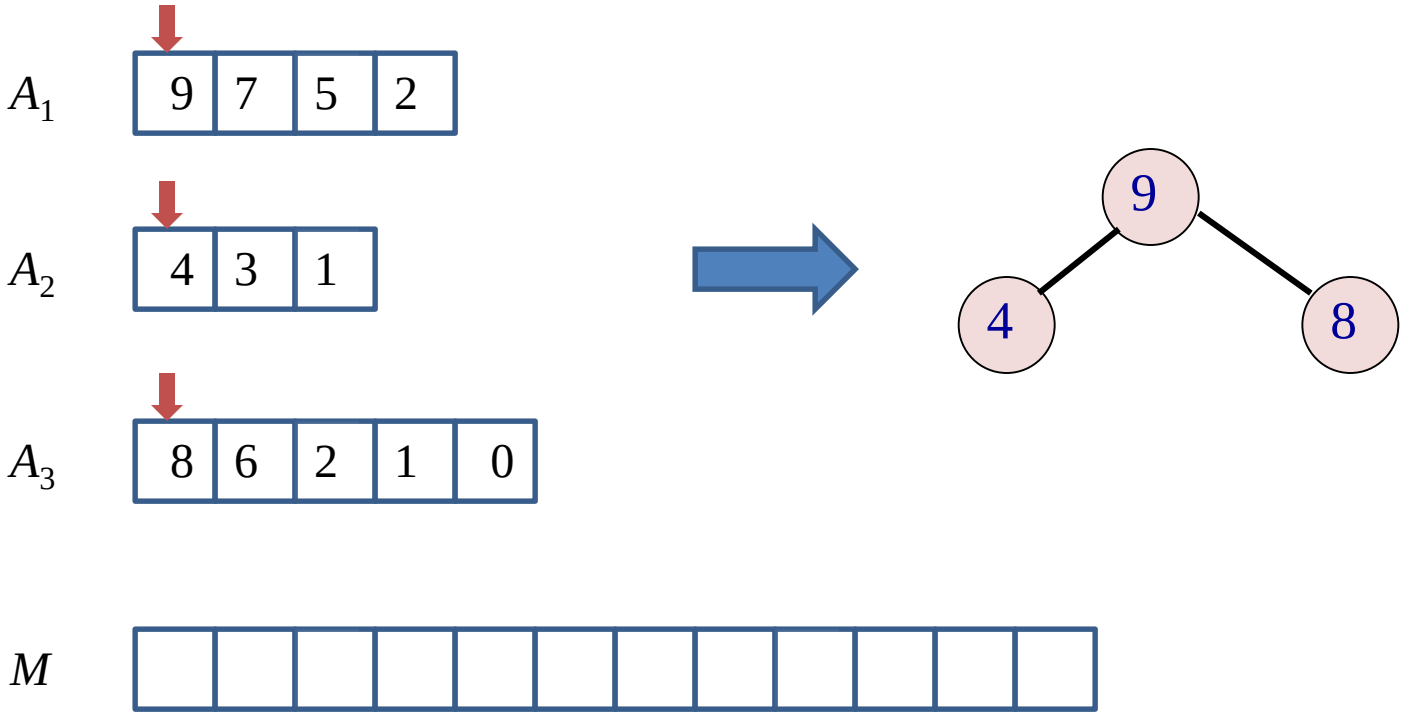


Question 9

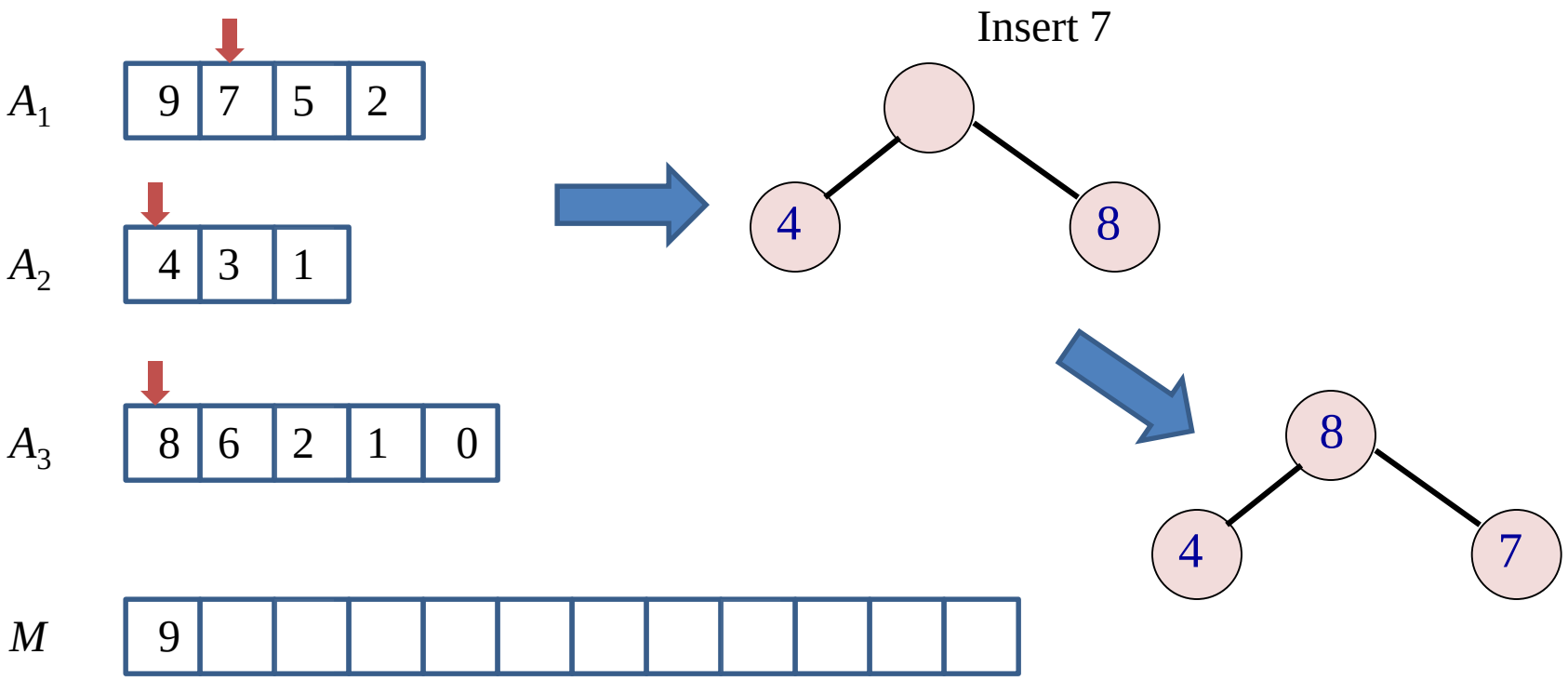
- Given k lists with a total of n numbers, where $k \geq 2$ and each list has been sorted in decreasing order, design an algorithm to merge the k lists into one list sorted in decreasing order, in running time $O(n\log_2 k)$.
- **Ans**: One algorithm is to use a maximizing heap:
  - First, insert the ***k*** elements from the heads of the ***k*** sorted lists into a maximizing heap. This will take running time $O(k)$.
  - Then, use getMax() to obtain the maximum element from the heap and append it at the end the merged list (initially empty). Suppose this merged element was originally from the jth list. Then, insert the element from the *next* position of the jth list into the heap using fixHeap.
  - Continue in this fashion until all the k lists have been exhausted.

  Analysis: For one element, an insertion using fixHeap takes time $O(\log_2 k)$, hence for the n elements the running time is $O(n\log_2 k)$.

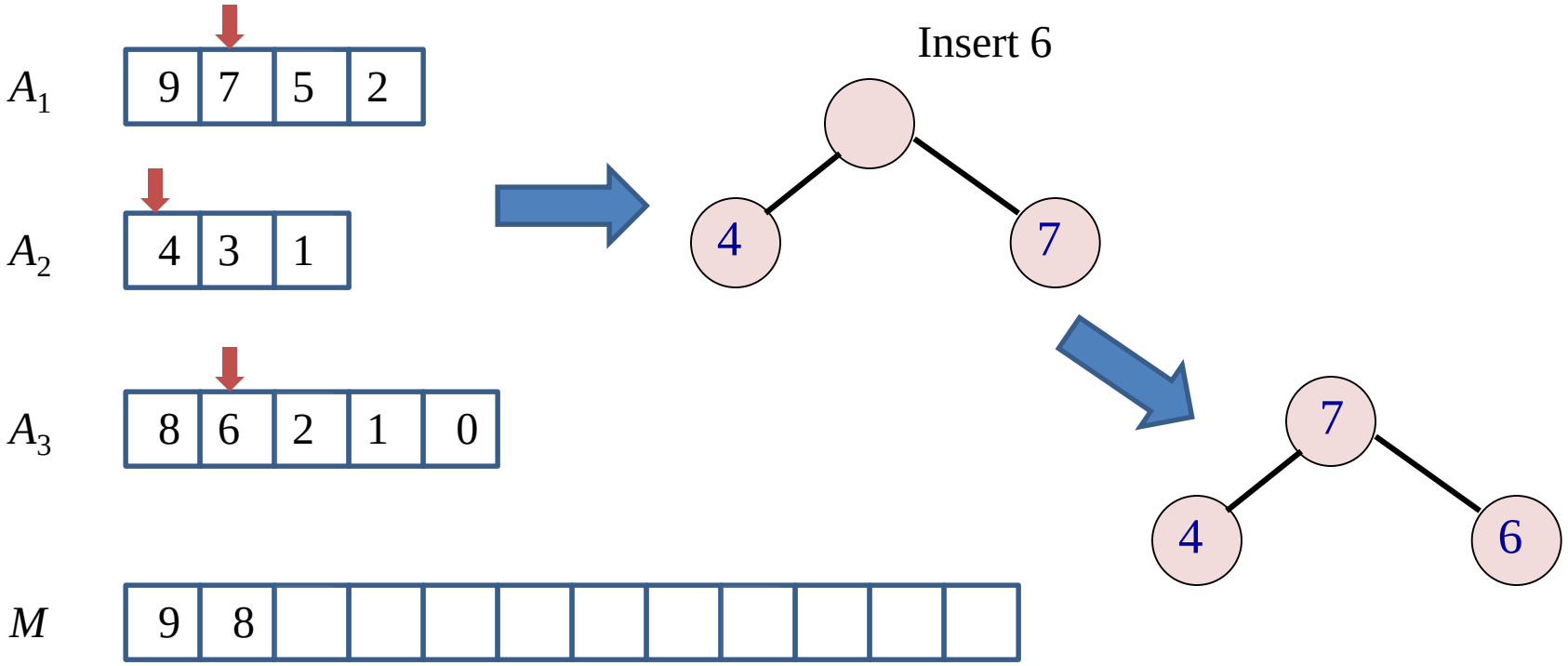
- Example:
  - Input: 3 sorted arrays
  - Output: A merged list *M*



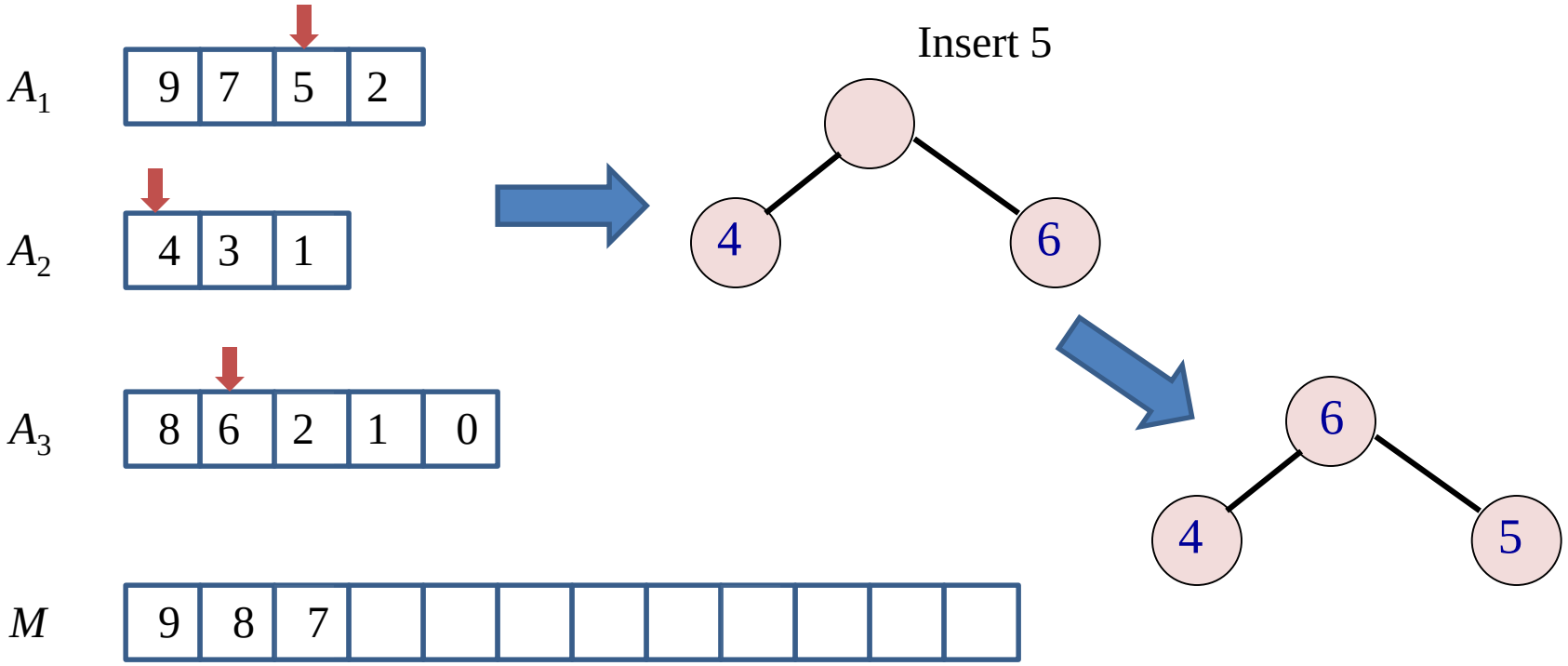
- Use getMax() to obtain the maximum element from the heap and append it at the end the merged list. Suppose this merged element was originally from the *j*th list. Then, insert the element at the *next* position of the *j*th list into the heap.



- Use getMax() to obtain the maximum element from the heap and append it at the end the merged list. Suppose this merged element was originally from the *j*th list. Then, insert the element at the *next* position of the *j*th list into the heap.



- Use getMax() to obtain the maximum element from the heap and append it at the end the merged list. Suppose this merged element was originally from the *j*th list. Then, insert the element at the *next* position of the *j*th list into the heap.



Continue in this fashion until all the input lists are exhausted…

What we have exercised

- Heapsort
  - constructHeap() O(n)
    - Heapifying() O(n)
      - fixHeap() O(lgn)
  - deleteMax() O(lgn)
    - fixHeap()
  - Complexity analysis
    - Counting the number of comparisons and swaps
    - Asymptotic complexity of Heapsort: O(nlgn)
- Algorithm Design
  - Use sorting algorithms
  - Familiar with each algorithm and its time complexity