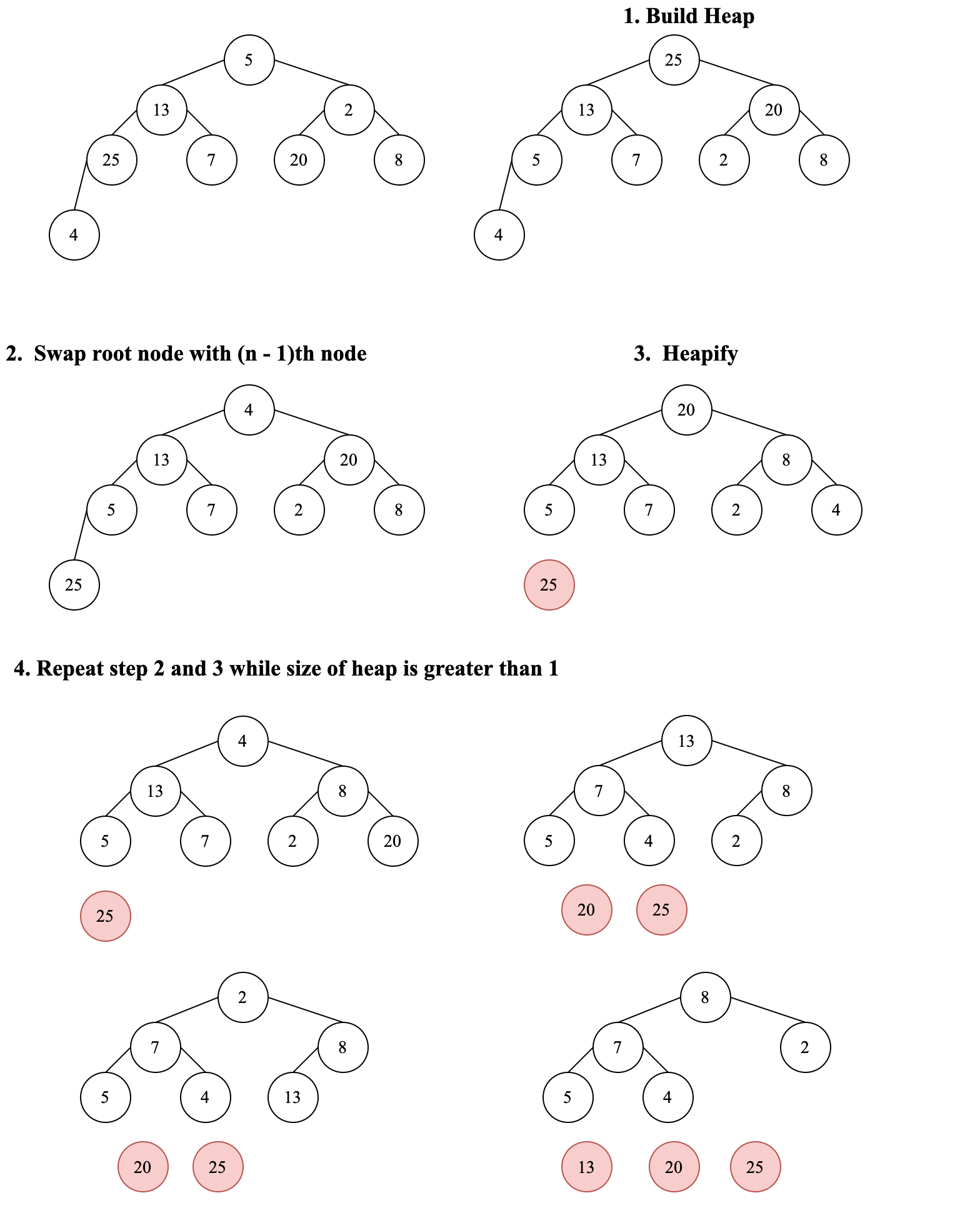
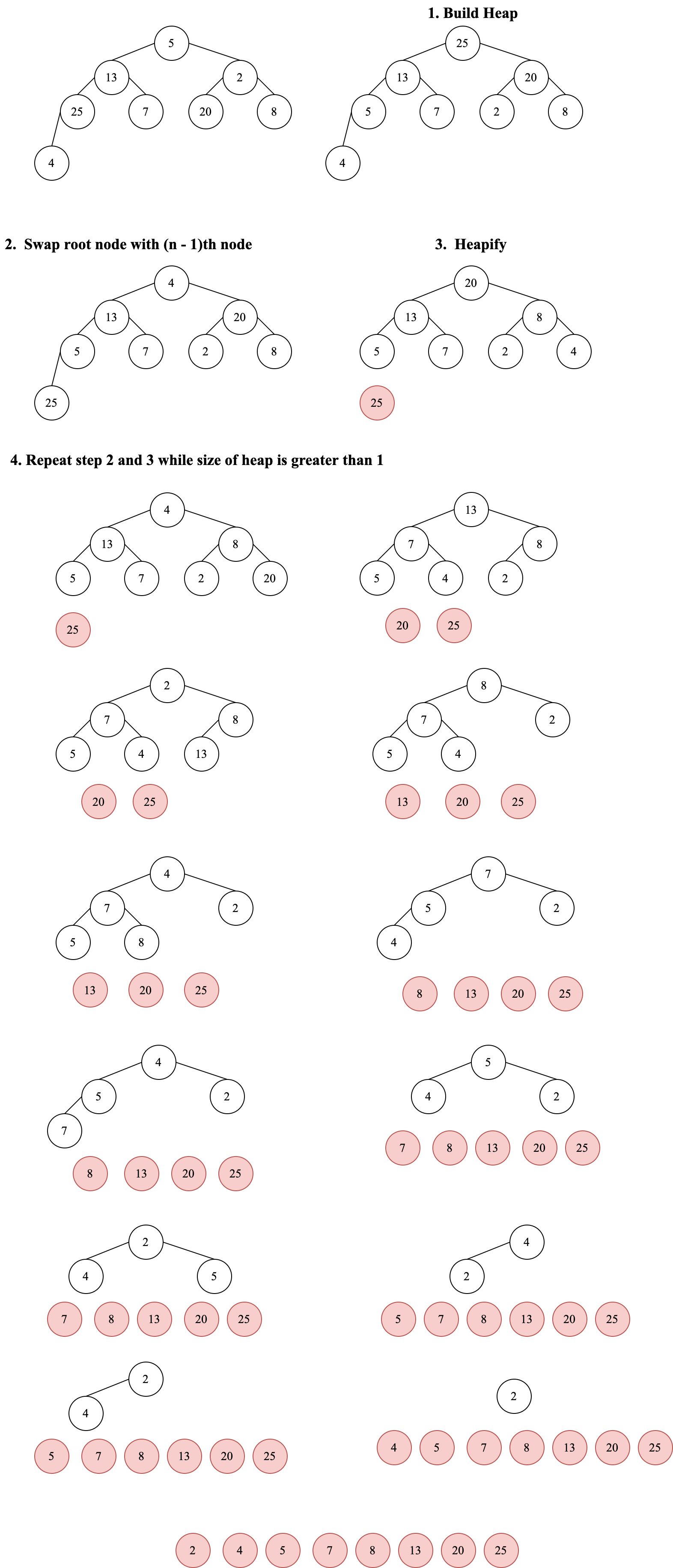
|  |
| --- |
| **CHAPTER 2 HOMEWORK** |

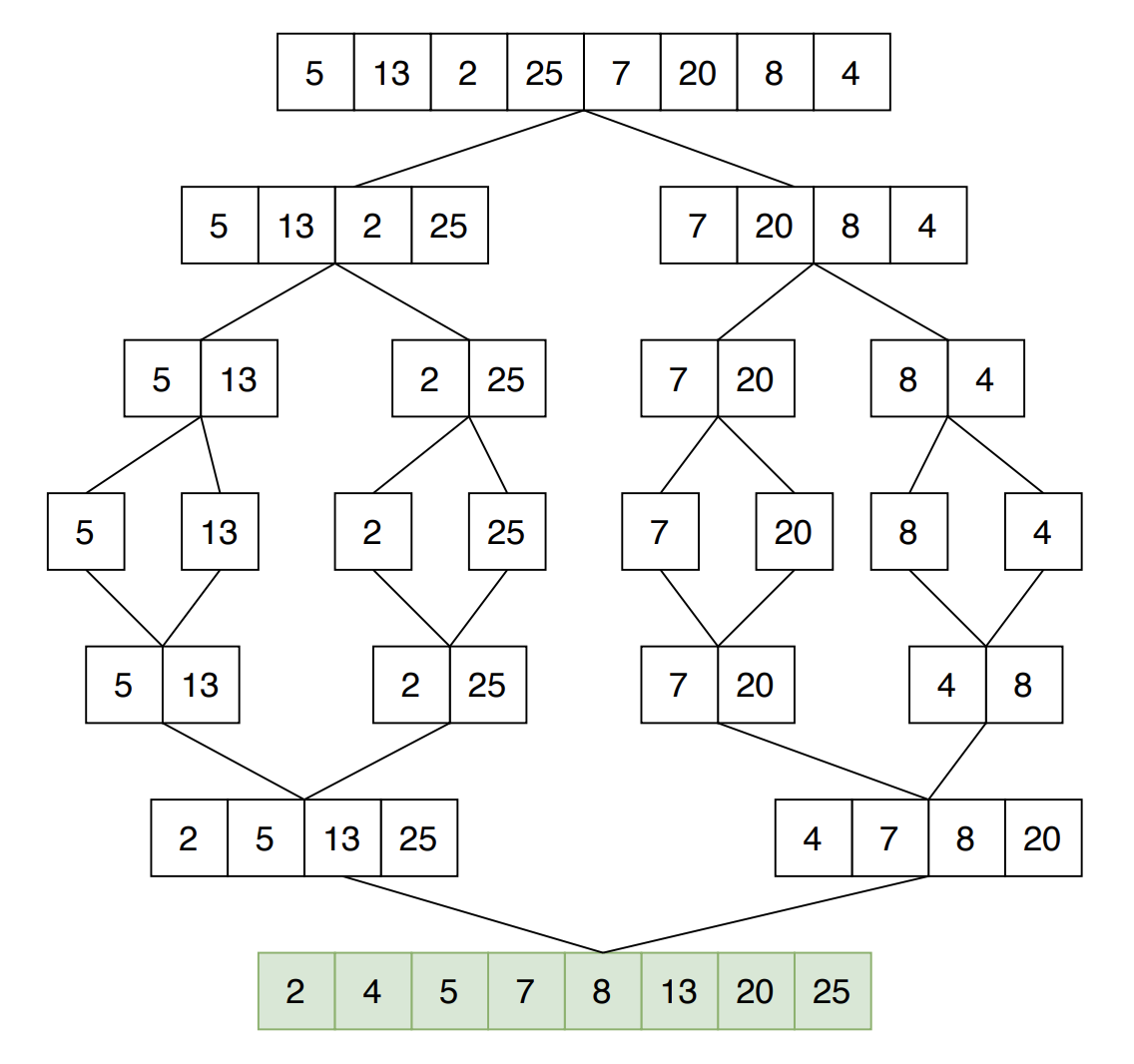
**PROBLEM 1:**

1. **𝐴 = < 5, 13, 2, 25, 7, 20, 8, 4 >**

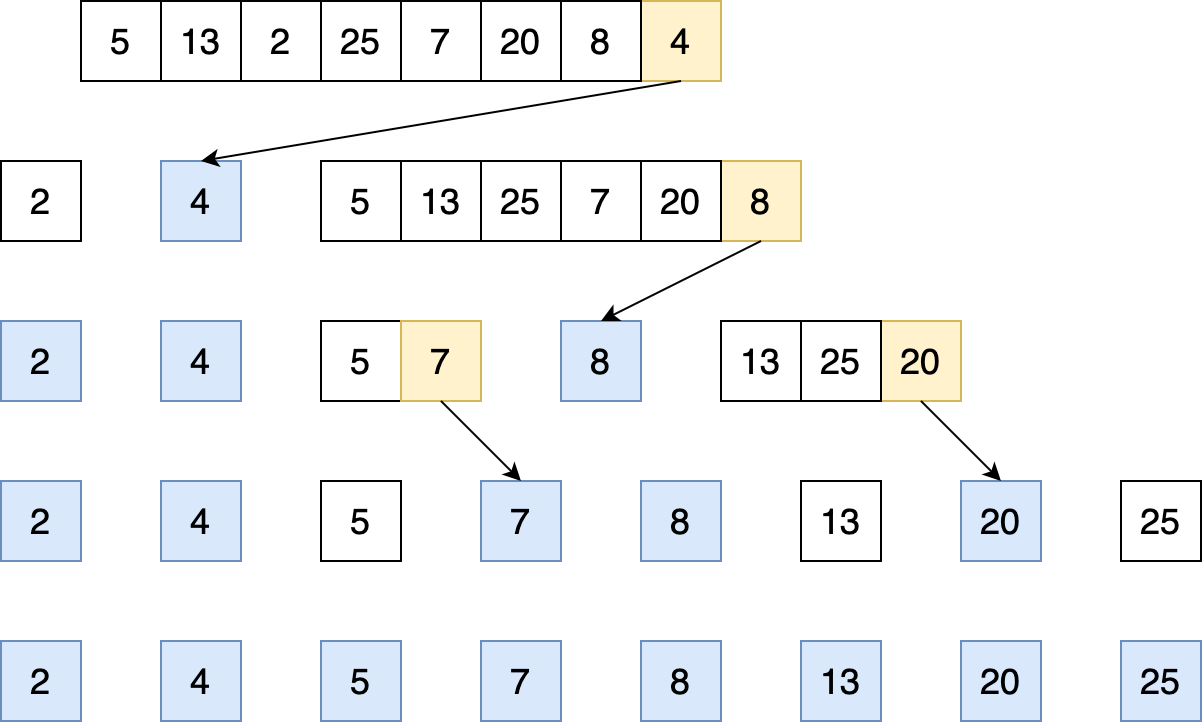
**HEAP SORT**

****

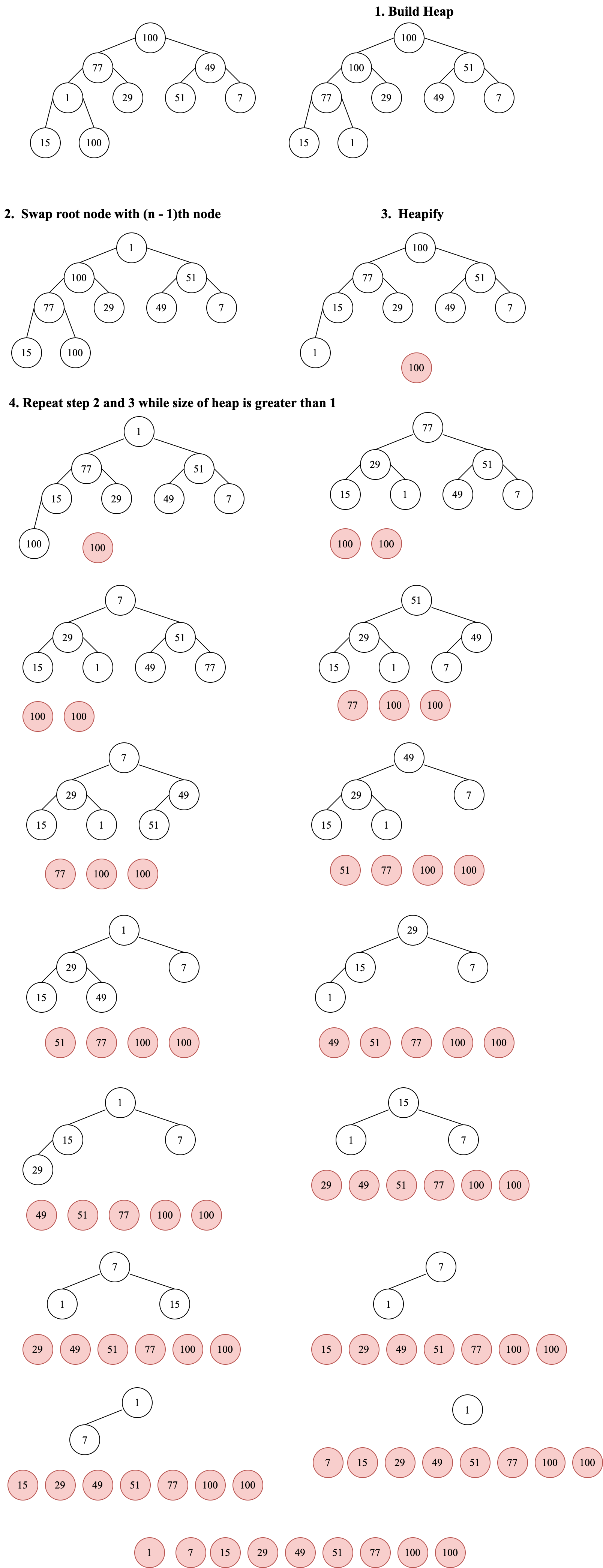
****

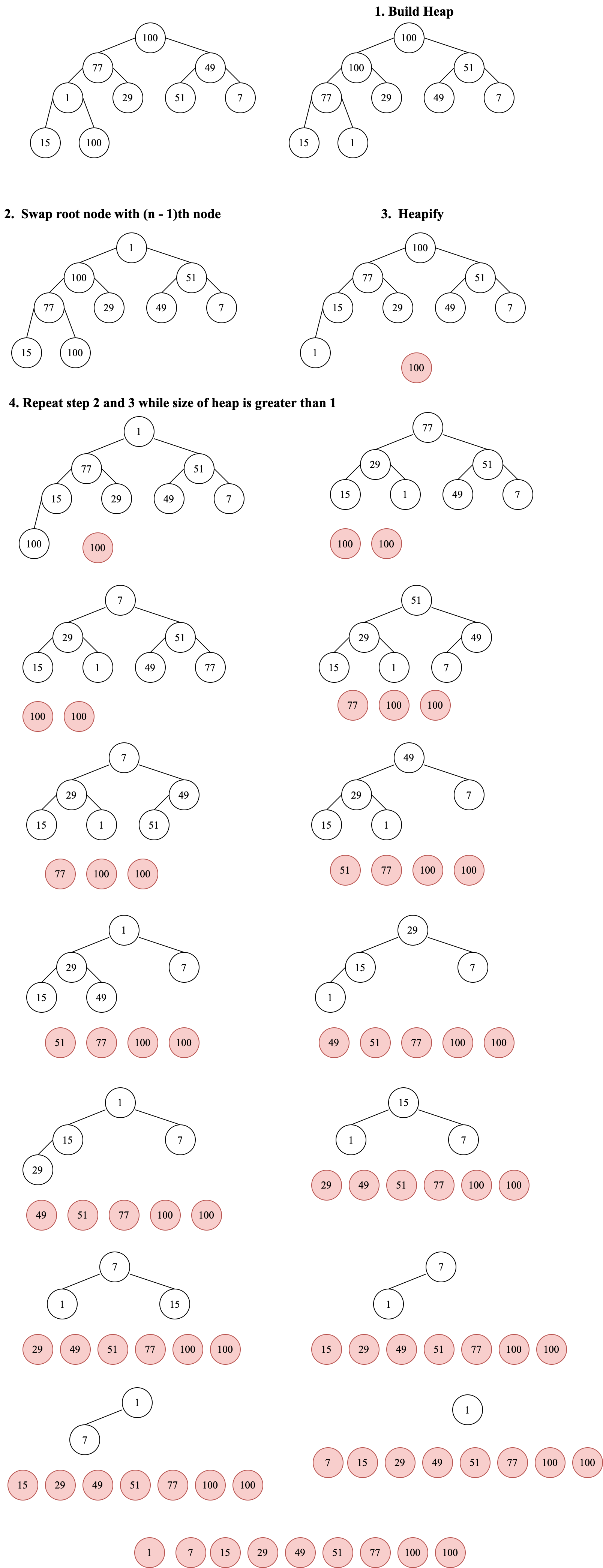
**MERGE SORT**

**QUICK SORT**

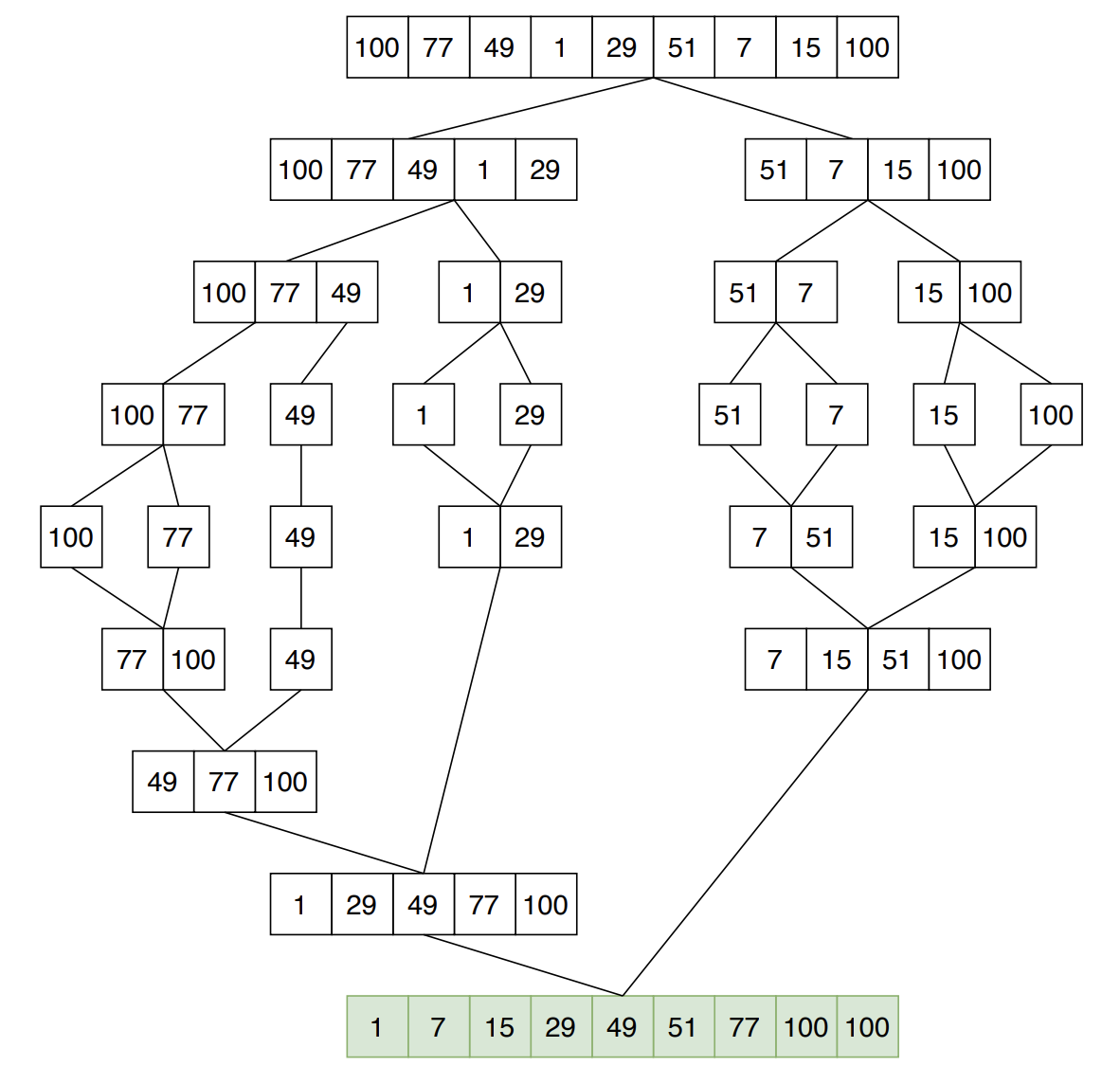


1. **𝐵 = < 100, 77, 49, 1, 29, 51, 7, 15, 100 >**

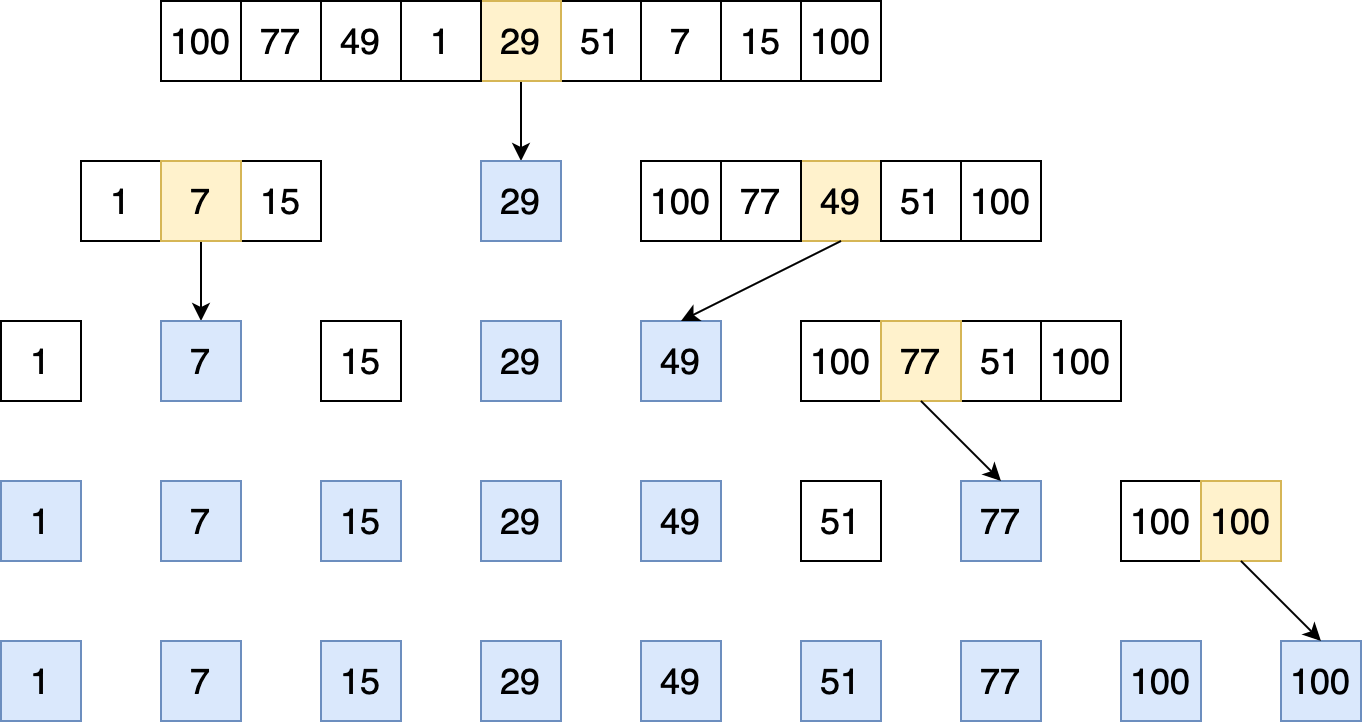
**HEAP SORT**



**MERGE SORT**

****

**QUICK SORT**

****

**PROBLEM 2:**

Gọi k là số con bắt đầu từ 1 node, vì là heap nên k = 2

* Số phần tử trong 1 heap có chiều cao h = 0 là 1
* Số phần tử trong 1 heap có chiều cao h = 1 là 1 + k = 3
* Số phần tử trong 1 heap có chiều cao h = 2 là 1 + k + k2 = 7
* …..
* Số phần tử trong 1 heap có chiều cao h = n là 1 + k + k2 +… + kn  = = 2*h*+1 - 1

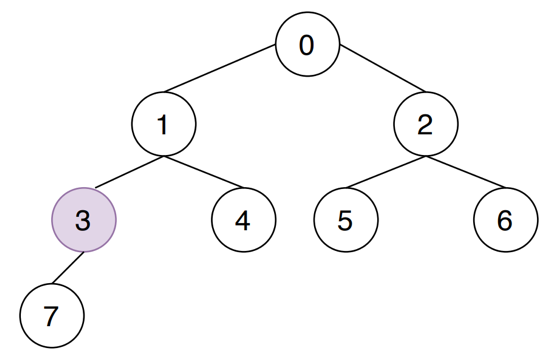
Số phần tử ít nhất trong 1 heap có chiều cao k là **2h**

Số phần tử nhiều nhất trong 1 heap có chiều cao k là **2*h*+1 - 1**

**PROBLEM 3:**

* Gọi h là chiều cao của heap.
* Heap là 1 cây nhị phân gần như hoàn chỉnh Cây con có chiều cao h – 1 là một cây nhị phân hoàn chỉnh.
* Heap có ít nhất 2h phần tử và có nhiều nhất là 2*h*+1 – 1 (ở PROBLEM 2)
* Vì h là số nguyên nên h = log2 𝑛

**PROBLEM 4:** Giả sử ta có mảng với n = 8



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Node cha cuối cùng là 3 ở vị trí thứ ⌊𝑛/2⌋ 1=> Các node lá phải ở vị trí thứ ⌊n/2⌋, ⌊𝑛/2⌋ + 1, … , 𝑛 − 1.

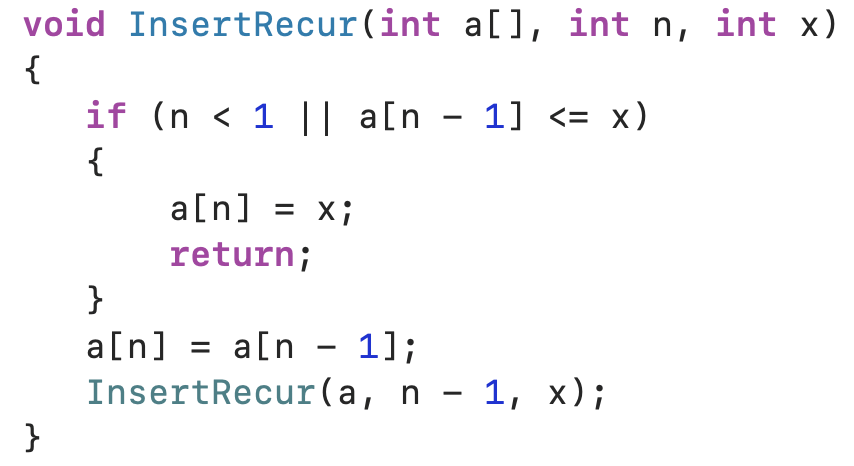
**PROBLEM 5:**

Thời gian thực hiện của Heapsort trên một mảng đã được xếp tăng dần và giảm dần đều là **O (nlogn)**.

**PROBLEM 6:**

Thời gian chạy của Quicksort khi tất cả các phần tử trong mảng đều có giá trị giống nhau là **O (n2)** : trường hợp xấu nhất.

**PROBLEM 7:**



**PROBLEM 8:**

* Giả sử ta có 2 mảng con lần lượt là:

a1 = [2 5 13 25] n1 = 4

a2 = [4 7 8 20] n2 = 4

* Mảng lưu kết quả khi gộp : sort\_arr = []
* Khởi tạo i = 0, j = 0 : chỉ số bắt đầu của a1 và a2

- a1[i] < a2[j] => chèn a1[i] vào cuối mảng sort\_arr, tăng i lên 1 đơn vị

**=> sort\_arr = [2], i = 1, j = 0**

- a1[i] > a2[j] => chèn a2[j] vào cuối mảng sort\_arr, tăng j lên 1 đơn vị

**=> sort\_arr = [2 4], i = 1, j = 1**

- a1[i] < a2[j] => chèn a1[i] vào cuối mảng sort\_arr, tăng i lên 1 đơn vị

**=> sort\_arr = [2 4 5], i = 2, j = 1**

- a1[i] > a2[j] => chèn a2[j] vào cuối mảng sort\_arr, tăng j lên 1 đơn vị

**=> sort\_arr = [2 4 5 7], i = 2, j = 2**

- a1[i] > a2[j] => chèn a2[j] vào cuối mảng sort\_arr, tăng j lên 1 đơn vị

**=> sort\_arr = [2 4 5 7 8], i = 2, j = 3**

- a1[i] < a2[j] => chèn a1[i] vào cuối mảng sort\_arr, tăng i lên 1 đơn vị

**=> sort\_arr = [2 4 5 7 8 13], i = 3, j = 3**

- a1[i] > a2[j] => chèn a2[j] vào cuối mảng sort\_arr, tăng j lên 1 đơn vị

**=> sort\_arr = [2 4 5 7 8 13 20], i = 3, j = 4**

- Do j >= n2, khi tăng i đến khi i < n1 đồng thời thêm phần tử ở a1[i] vào sort\_arr. Ta được mảng đã sắp xếp là **sort\_arr = [2 4 5 7 8 13 20 25]**

**PROBLEM 9:**

