Q1.

```
/**
 * A BST node.
 */
struct BNode {
  int key;
  BNode *left;
  BNode *right;
};

static std::vector<int> keys;
```

```
struct ANode {          // A Node in an AVL tree.
  int key;              // a word
  int height;           // height of the node
  ANode *left;          // pointer to the left child of the node (or NULL)
  ANode *right;         // pointer to the right child of the node (or NULL)
};

void dump_tree(BNode * r) {
  if( r == NULL ) return;
  dump_tree(r->left);
  keys.push_back(r->key);
  dump_tree(r->right);
}
```

```
ANode *build_avl(int start, int end, int d) {
  if (end <= start) return NULL;
  ANode *center = new ANode;
  center->left = build_avl(start, (start + end) / 2 - 1, d + 1);
  center->key = keys[(start + end) / 2];
  center->height = d;
  center->right = build_avl((start + end) / 2 + 1, end, d + 1);
  return center;
}

ANode *bst2avl(BNode *root) {
  dump_tree(root);
  return build_avl(0, keys.size(), 0);
}
```

Assignment #3 e8y0b

2. The process to split a treap into two treaps involves two procedures: (i) removing the left most node from the treap if it's smaller than the given key

(ii) inserting the node to the new treap and ensuring the treap property can be maintained. From Project #2, we know that by changing the priority of the root node to infinity and swaping it down to a leaf, we can remove it while keeping the heap property. The second procedure works pretty similar to the insert function (we do not want to keep the original priority.) We insert the root node as a leaf, then swap it up to the appropriate position.

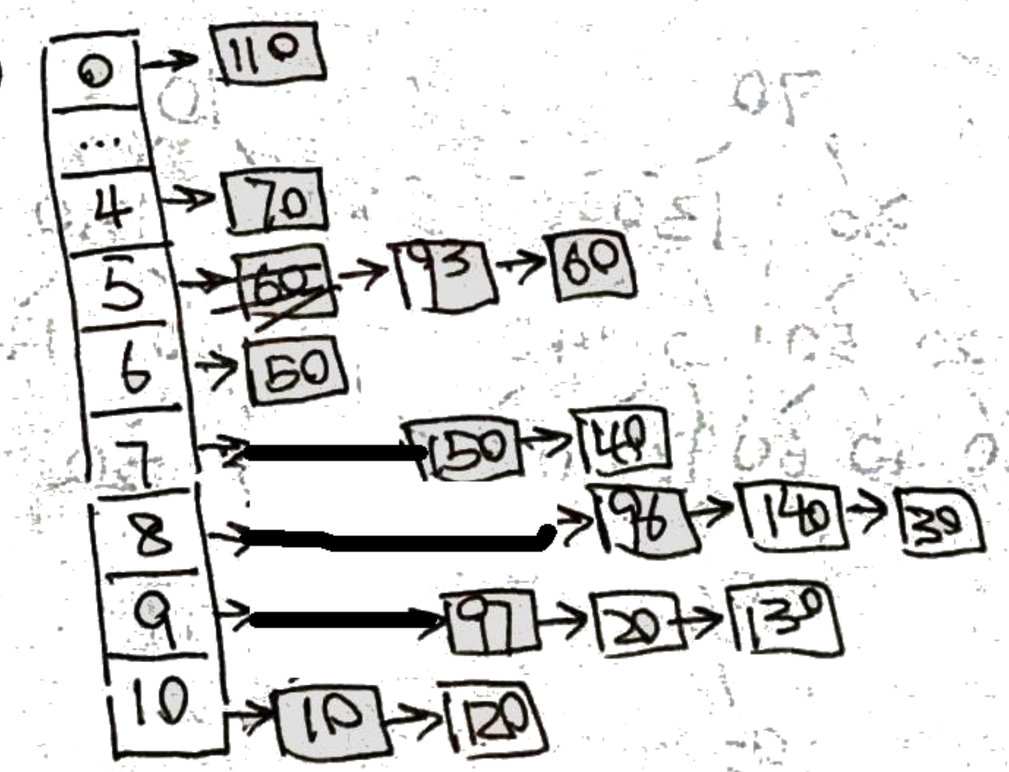
When the left most node of the original heap has a value greater than the key the process is finished.

After we remove the left-most node from the treap, we need to link its parent node to the right-side node

Since the left most position is already a leaf, we only need to consider the procedure after removing it.

A few thoughts:
we do not want to keep the original priority because it does not control the height of the new treap. We're very likely to remove the entire left side of the original heap, which already it has the priority in sorted order.

3. (a)



3. b) 1 → 70, (1)

2 → 140, (2)

4 → 50, (4) < 5 → 120, 5

7 → 30, (7)

10 → 10, (1, 4, 7, 10)

12 → 150, (12)

14 → 60, (14) < 13 → 93, (1, 4, 7, 10, 13)

15 → 130, (15) < 16 → 96, (14, 10, 16)

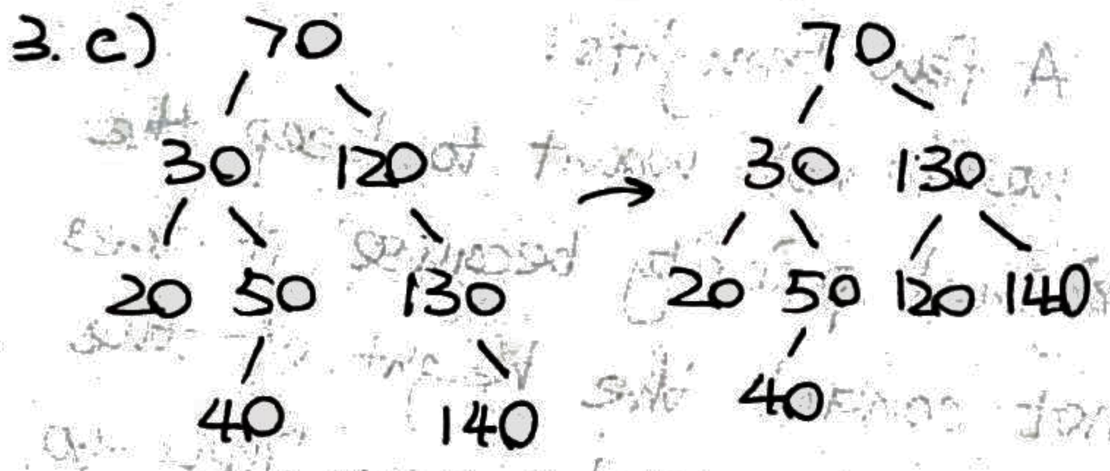
17 → 40, (17)

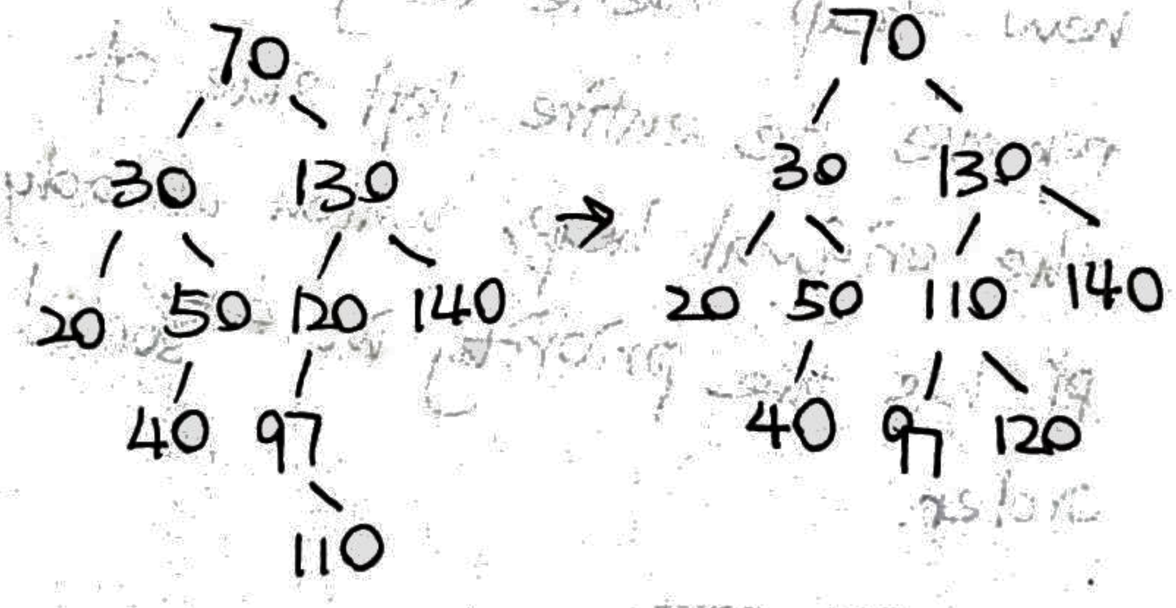
18 → 110, (18)

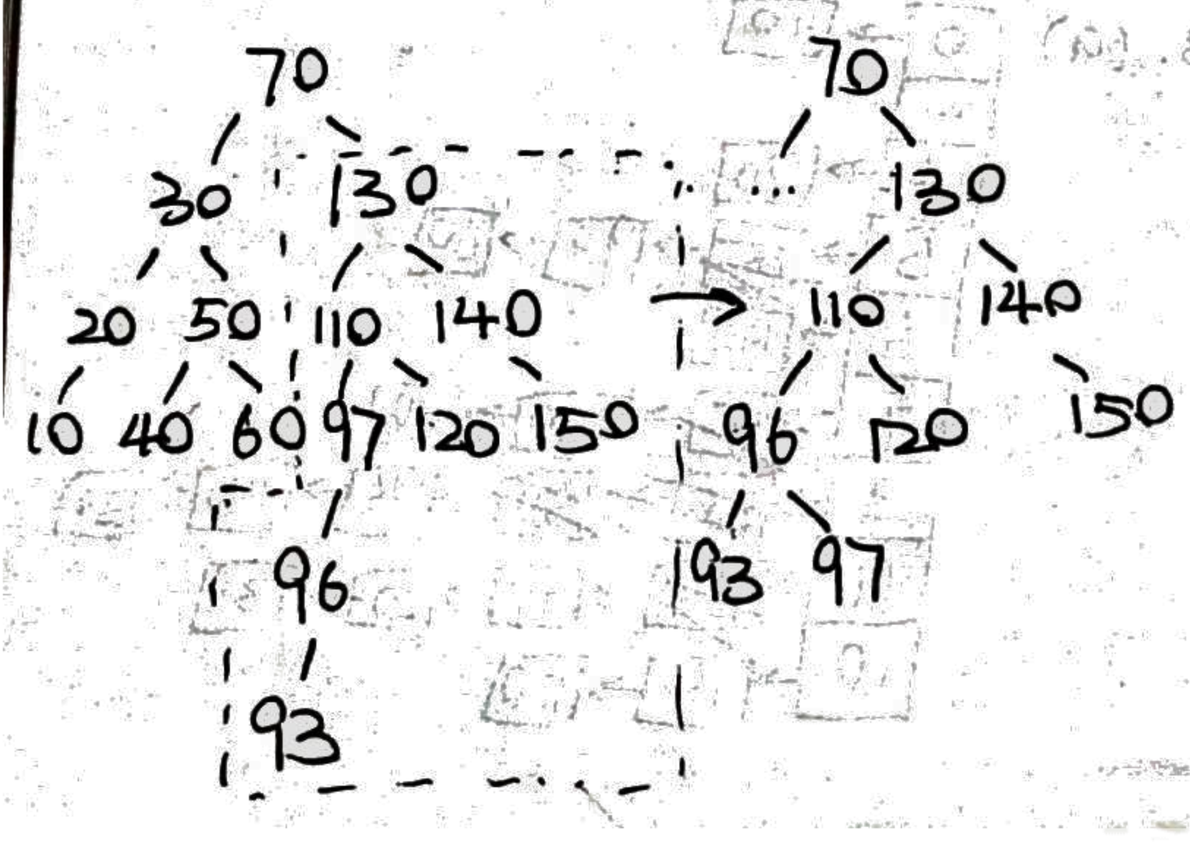
19 → 97, (5, 12, 19)

20 → 20, (20)

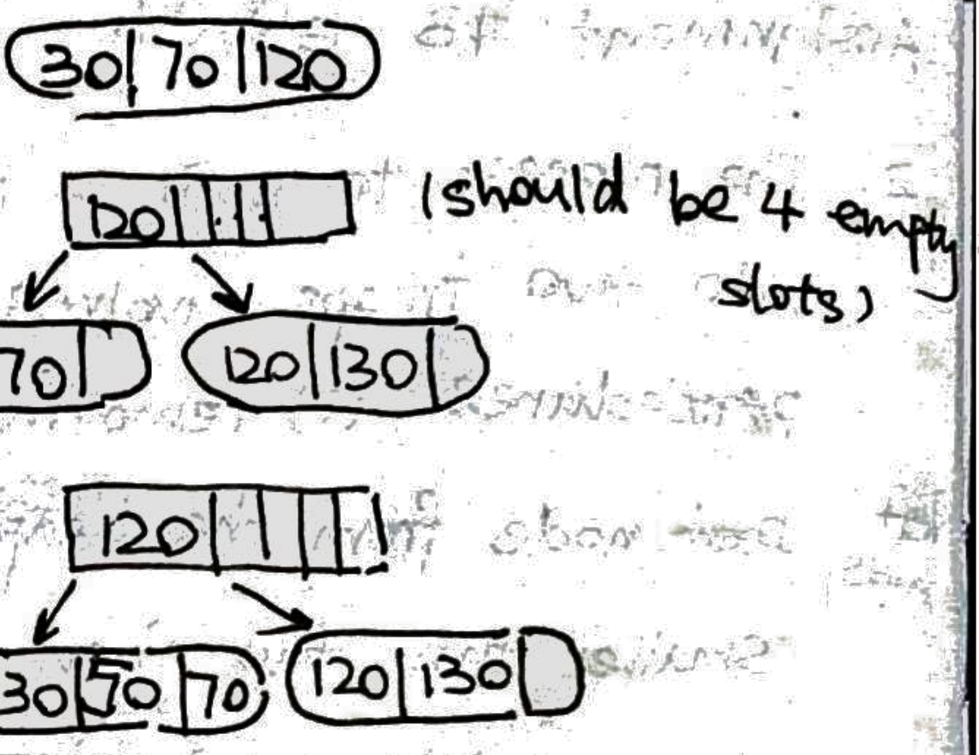
3. c)

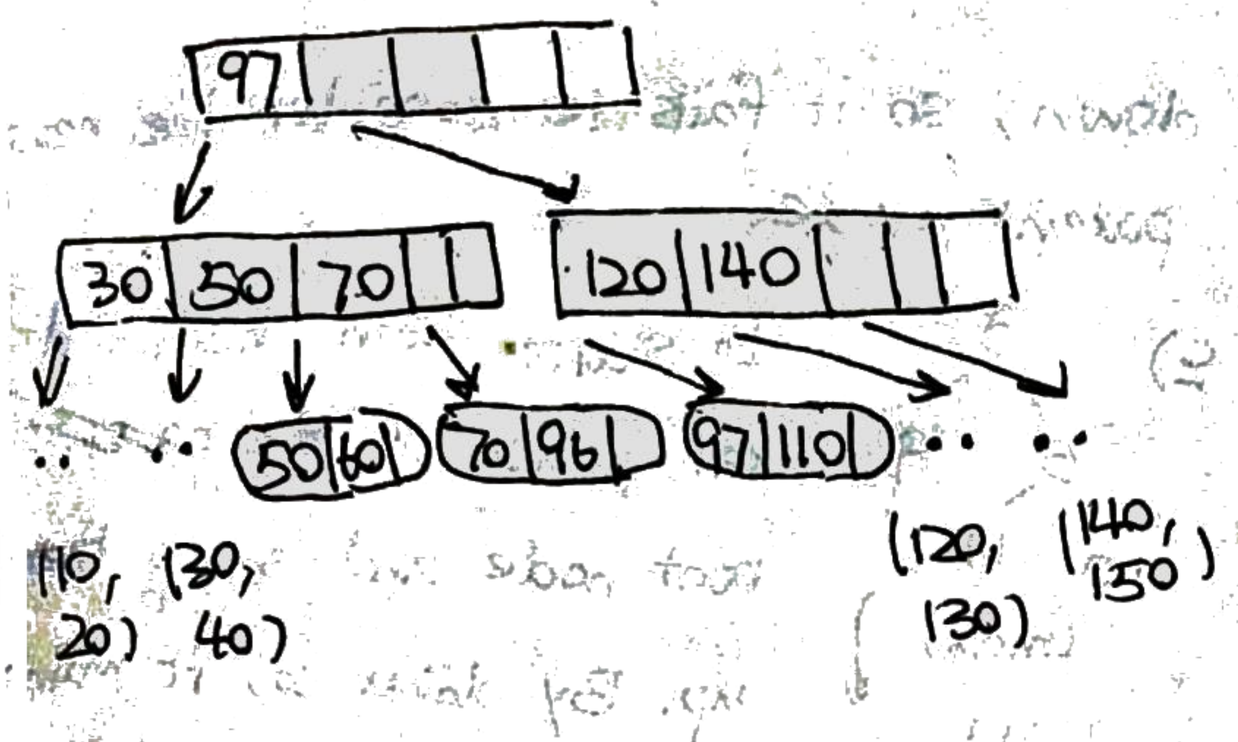






3. d)





4.(a) $(S_0 + S_1 \cdot 256 + S_2 \cdot 256^2 + S_3 \cdot 256^3) \bmod (256^2 - 1)$

$\equiv (S_0 + S_1 \cdot 256 + S_2 + S_3 \cdot 256) \bmod (256^2 - 1)$

$\equiv (S_0 + S_2 + (S_1 + S_3) \cdot 256) \bmod (256^2 - 1)$

Therefore, if the sum of S_0 and S_2 AND the sum of S_1 and S_3 do not change, the new string would be allocated to the same slot. In order to count the number of strings we can observe the two characters case:

① 0d 0D → 0D 0 1 → 0101 → 257

② 0 2 00 → 00 02 → 513

③ 0 3 01 → 01 03 → 769

0 3 00 → 00 03

FF 00 ... → 65280

FF FE → FE FF → 65279

Total # two characters collisions

Since all of the characters are smaller than 256, (S_0, S_2) and (S_1, S_3) can be considered as independent pairs.

$(2+256) \cdot 256/2 + (2+256) \cdot 254/2$

$= 65538$

4.(b)

$S_0 + S_1 \cdot 256 + S_2 \cdot 256^2 - (S_0 \cdot 256^2 + S_1 \cdot 256 + S_2)$

$= (256^2 - 1) S_0 + (256^2 - 1) S_1$

Since the difference between the hash value of these two strings are divisible by $256^2 - 1$, they would be placed at the same slot.

4.(c) $h(S_0 S_1 \ldots S_{10}) - h(S_0' S_1' \ldots S_{10}') \bmod m$

$\equiv (S_0 - S_0') + 256 \cdot (S_1 - S_1') + (S_2 - S_2') + 256 (S_3 - S_3') + \ldots + 256 (S_9 - S_9') + (S_{10} - S_{10}') \bmod m$

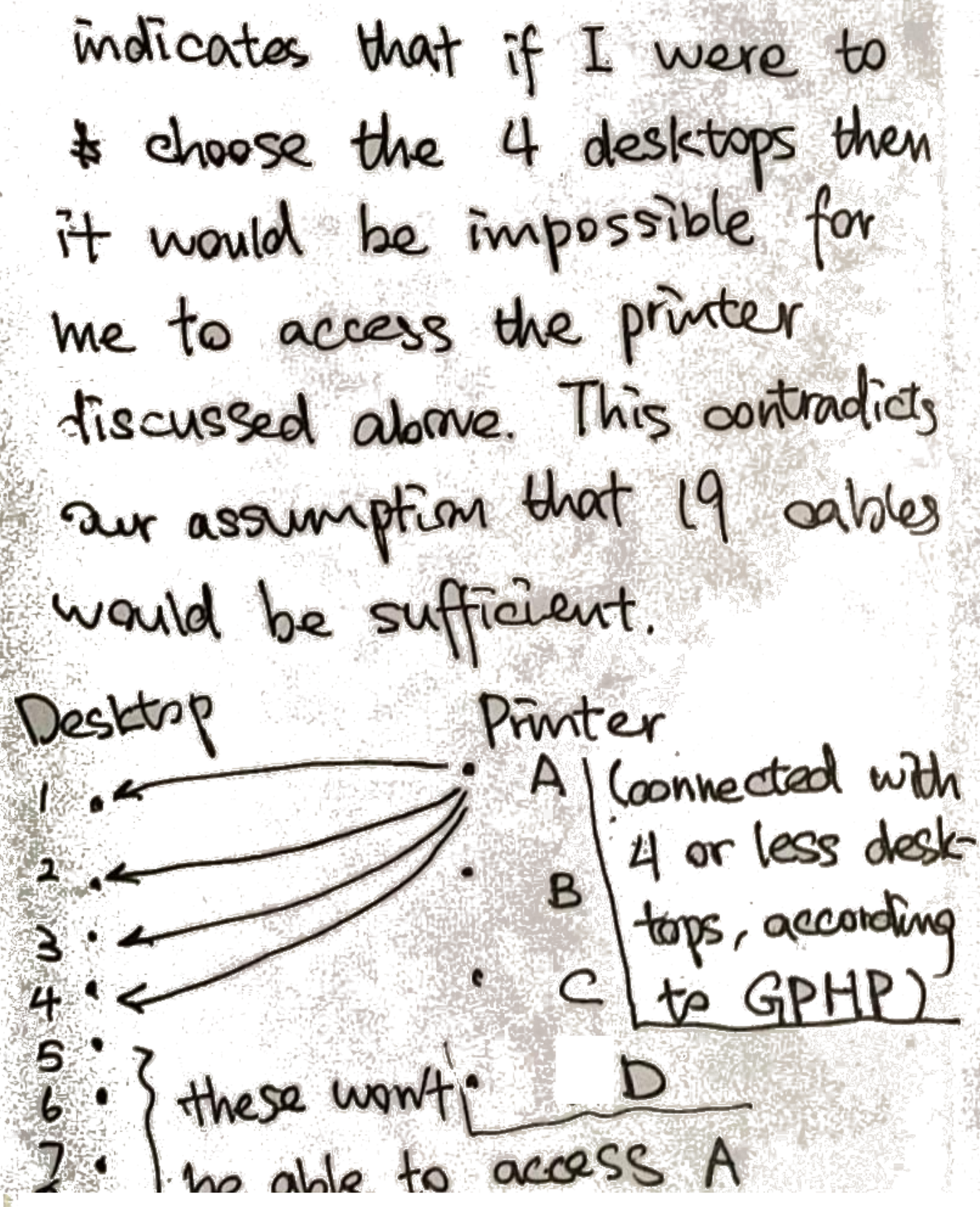
$= (S_0 + S_2 + S_4 + S_6 + S_8 + S_{10}) - (S_0' + S_2' + S_4' + S_6' + S_8' + S_{10}') + 256 \cdot [(S_1 + S_3 + S_5 + S_7 + S_9) - (S_1' + S_3' + S_5' + S_7' + S_9')] \bmod m$

From the above equation we can observe that the ten-characters strings keep if the single bits and the sum of the double bits the same then the new string would be in the same slot.

5. Minimum number : 20

Proof by contradiction:

suppose 19 cables would be enough. Since there are 4 printers, there exists are printer that cannot connect to desktops at least 4 ($\lfloor 19/4 \rfloor = 4$, $8-4=4$) This indicates that if I were to * choose the 4 desktops then it would be impossible for me to access the printer discussed above. This contradicts our assumption that 19 cables would be sufficient.



6. a)

Create a queue that maintains the status of all the nodes in the heap. Every five push all of the leaves into the queue and perform swap up parallelly. If a node has two siblings, wait the threads that operate on them to finish first. When (cutoff case) all of the leaves are placed at the right spots, the program terminates.

b) Since the span of the quick sort algorithm is $\lg n \lg n$, replacing the compare function pivot of the should give us a good result.

```
A short description of the algorithm:
Partition the array into two parts with
a pivot. If the index of the pivot
equals the given k, then return the
pivot (we've found the k-th largest
element); if it is larger than k, then
fork a process to work on the left
side of the index recursively; if
smaller, then work on the right side
and find the (k - (index of pivot))th
element.
```