- Shell sort:
  + Initialize the value of h (gap)
  + Divide the list into smaller sub-list of equal interval h
  + Sort these sub-lists using insertion sort
  + Decrease value of h and repeat the shell sort

* selection
- ~~Bubble~~ sort
  + Start from the first element of array, find the smallest element in the array and swap them
  + move to the next element and repeat the code.

~~+~~ Bubble sort:
  + Compare a pair of nodes starting from the first node of the array.
    - If the first node in the pair greater than the second node, swap them
    + else move on to the next element in the array and compare again until finish the array
  + Repeat the code again until all element are sorted.

~~+~~ Insertion sort:
  + Start from first element of array, compare to the next element, swap if the second element less than first one.
  + Compare them to the next element and repeat until finish the array.

= Quick Sort:
  ① + choose the highest index value has pivot
  ② + Take 2 pointer point to left and right of the list excluding pivot
  ③ + left point to the low index, right point to the high
  ④ + While value at left is less than pivot move right
  ⑤ + while value at right is greater than pivot move left.
  ⑥ + If step 5 doesn't match, swap left & right
  ⑦ + if left ≥ right, the point where they met is new pivot

1. (20 points) **Sorting Algorithms**: Define the following sorting algorithms using text and or pseudo code.

a. Selection Sort

- start from the first element of array, find the smallest element in the array and swap ~~them~~
- move to the next element and repeat the code

```
int i, min;
for (i = 0; i < size, i++){
    min = find_min_index (a, size, i);
    swap (&a[i], &a[min]);
}
```

+5

b. (5 points) Bubble Sort

- Compare a pair of nodes starting from the first node
  - if the first node in the pair greater than the second node, then swap
  - else move on next element in the array
- repeat again

```
int i, j;
for (j = 0; j < size; j++){
    for (i = 0; i < size, i++){
        if (a[i] > a[i+1]){
            swap (&a[i], &a[i+1]);
        }
    }
}
```

+5

c. (5 points) Insertion Sort

```
int i, j;
for (i = 0; i < size; i++) {
    j = i;
    while (j-1 >= 0 && a[j] < a[j-1])
    {swap (&a[j], &a[j-1]);
     j--;
    }
}
```

+5

start from first elem of array, compare to the next element, swap if the second element less than first one. - compare them to the next elem and repeat until finish the array

d. (5 points) Shell Sort

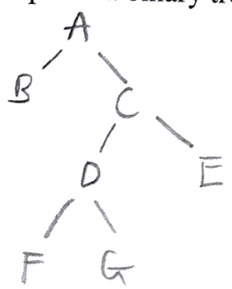
```
int i, j, h;
h = size/3 + 1;
do {
    h--;
    for(i = h; i < size; i++){
        j = i;
        while (j - h >= 0 && a[j] < a[j-1]){
            swap (&a[j], &a[j-1]);
            j--;
        }
    }
} while (h != 1);
```

+5

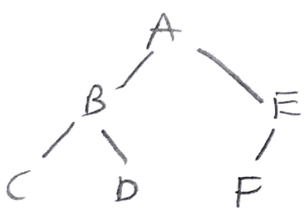
2. (12 points) **Full vs Complete Binary Trees**

12

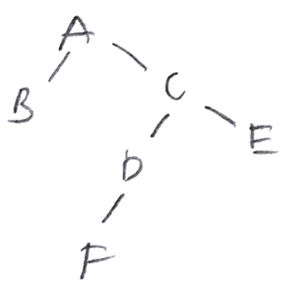
a. (3 points) Give an example of a binary tree that is full but not complete.



b. (3 points) Give an example of a binary tree that is complete but not full.



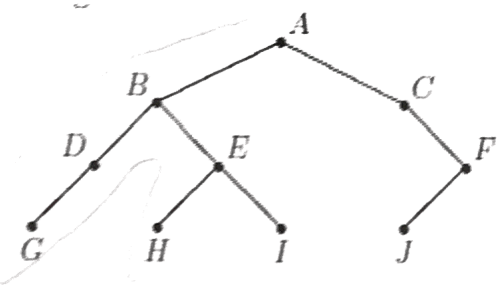
c. (3 points) Give an example of a binary tree that is neither full nor complete.



d. (3 points) Give an example of a binary tree that is both full and complete.

10

3. (15 points) List the sequence of nodes visited by pre-order, in-order, and post-order traversals of the following tree:



(a) (5 points) Give the output for a **preorder** traversal calling visit.

-2 A B D G H E I C F J.

E H I

(b) (5 points) Give the output for an **inorder** traversal calling visit.

G D B H I E J F C A

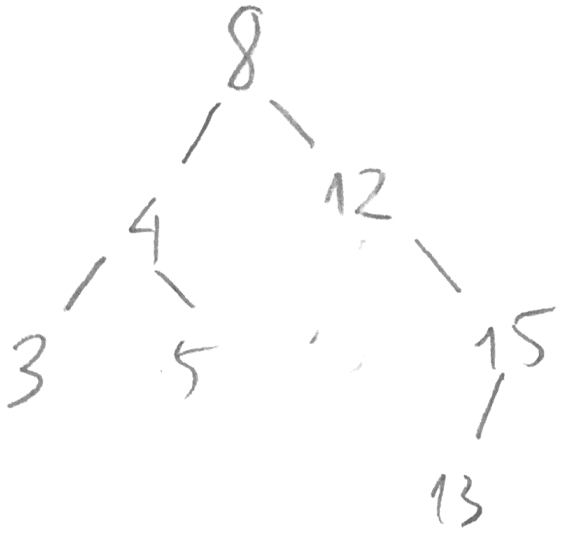
-3 E I A C J F

(c) (5 points) Give the output for a **postorder** traversal calling visit

G D H I E B J F C A.

4. (5 points) Assuming the following values arrive in the order they appear and are inserted into a ***binary search tree***, show the resulting tree.

8, 12, 4, 15, 3, 5, 13



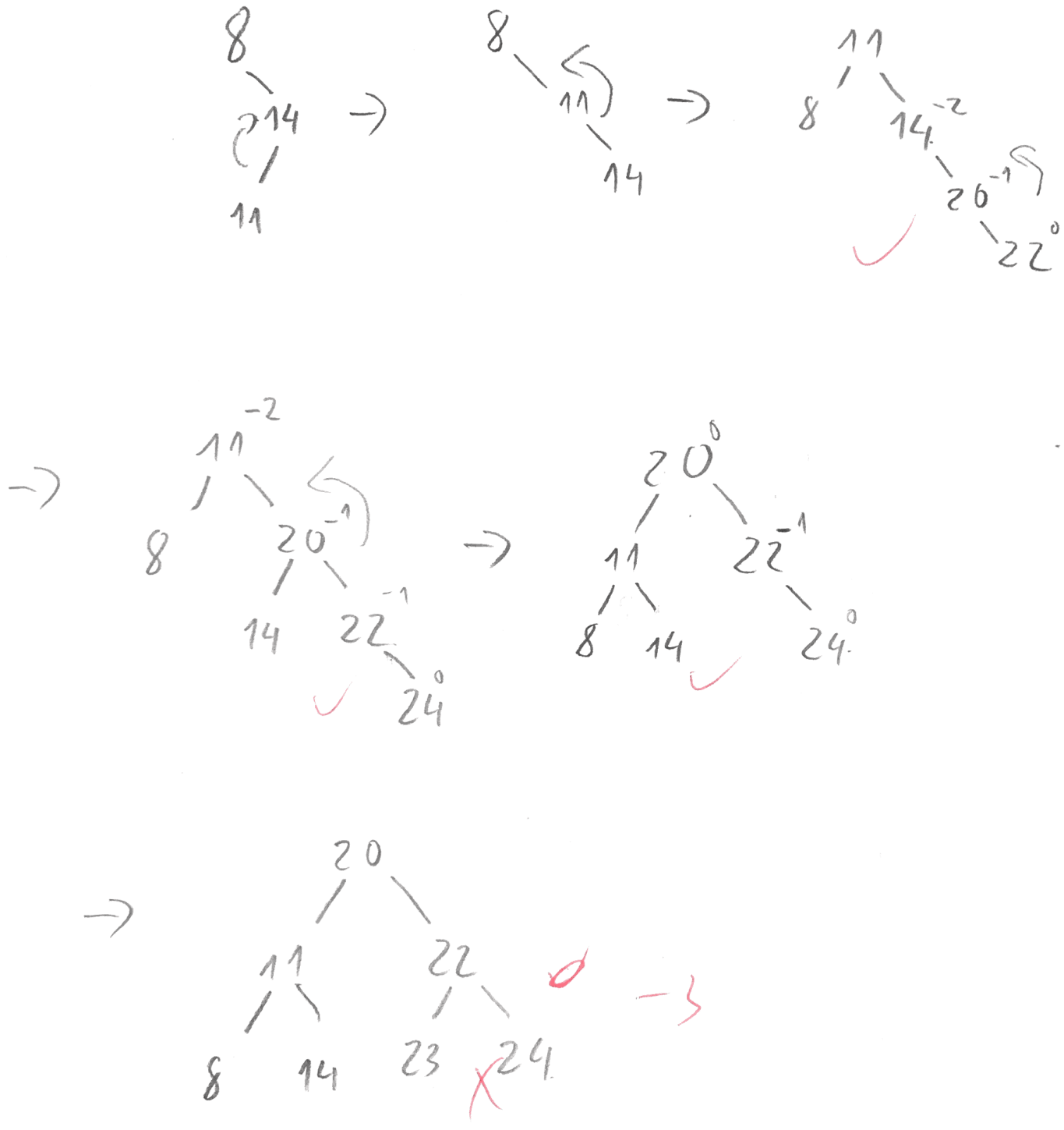
5. (10 points) Write down a ***recursive function to destroy a tree.*** The function takes a node pointer. The recursive function declaration is:
`void binary_tree_destroy(BinaryNode pNode);`

```
void binary_tree_destroy (BinaryNode *pNode) {
    if (pNode != NULL) {
        binary_tree_destroy ( &(pNode) -> left);
        binary_tree_destroy ( &(pNode) -> right);

        free (pNode);
        pNode = NULL;
    }
}
```

-5

6. (15 points) Assuming the following values arrive in the order they appear and are inserted into ***AVL tree***, show your work and the resulting tree.
8, 14, 11, 20, 22, 24, 23



7. (3 points) What would be the output of the following program:

3

```
int fun(int n)
{
  if (n == 4)
        return n;
  else return 2*fun(n+1);
}
int main()
{
  printf("%d ", fun(2));
  return 0;
}
```

16