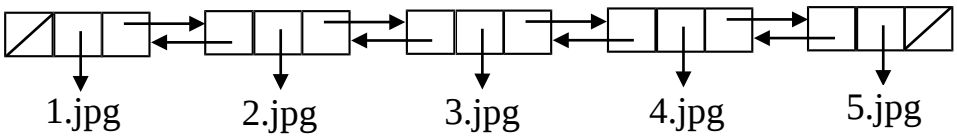
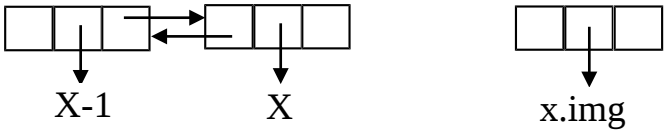
Student ID: 1422087

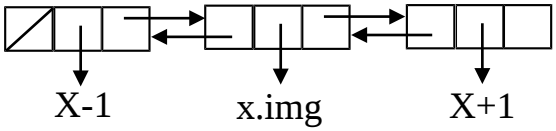
1. A double linked list diagram with 5 images would be something like this:



If the images are stored in a simple array and we want to insert an additional photograph x.jpg at a particular point (let's say that we need to store the image at point X), then we would have to change the structure of the array we have. The `previous` pointer (which points to the previous cell) of the image at point X should start pointing to the new image (and therefore the new image will have a new position X+1). Moreover, the `next` pointer of the image at position X-1 should point to the new picture. Finally, the `previous` pointer of the newly inserted image should point to X-1 and the `next` pointer should point to X+1. Before all the changes the situation is this:



And after all the changes in the pointers everything should like this:



In terms of time complexity, it is much faster to access elements in the array, since there every element has its position and we don't have to go through the whole array. In the case of doubly linked lists the time complexity is the same as normal lists – we need to go through the whole list if we want to access an element at a specific position.

2. If the positions in a list start from 0 (0,1,2,3 and so on), then the `getItem` procedure is this:

```
getItem(i,L) {
     if (isEmpty(L))
          error('Error: empty list in procedure getItem.')
     else if (i==0) return first(L)
     else getItem(i-1,rest(L))
}
```

3. Flip procedure:

```
flip(qt) {
     if (isValue(qt))
          return qt
     else return makeQT(flip(ru(qt)), flip(lu(qt)), flip(rl(qt)),
flip(ll(qt))
}
```

Avevalue procedure:

```
avevalue(qt) {
     if (isValue(qt)) {
          return qt
     }
     else return (avevalue(lu(qt))+avevalue(ru(qt))+
     avevalue(ll(qt))+avevalue(rl(qt)))/4
}
```

4. isLeaf procedure:

```
isLeaf(bt) {
     if (isempty(bt))
          return false
     else if((isempty(left(bt))) && (isempty(right(bt))))
          return true
     else return false
}
```

numLeaves procedure:

```
numLeaves(bt) {
     if (isempty(bt))
          return 0
     else if (isLeaf(bt))
          return 1
     else return 1+numLeaves(left(bt))+numLeaves(right(bt))
}
```

5. equalBinTree procedure:

```
equalBinTree(t1,t2) {
     if (isempty(t1) && isempty(t2))
          return true
     else if (root(t1) == root(t2)){
          return ( equalBinTree(left(t1),left(t2)) &&
equalBinTree(right(t1),right(t2)) )
     }
     else return false
}
```

The time complexity of this algorithm is linear or O(n), where n is the length of the smaller tree (considering both trees aren't equal).