Part 1:

My Event class:

```
class Event {
    public int year;           // the year of the event
    public String description;   // the event description
    public Event[] next;
    public Event[] prev;
    public Event nextEqual; //chain all the same year
    public int pillar; //pillar height
    // constructor
    public Event(int iyear, String idescription)
    {
    year = iyear;
    description = idescription;
    pillar = 0;
    }
    // print method
    public String toString()
    {
       return String.valueOf(year) + " " + description;
    }
}
```

Explain: Use Event[] array next,prev to link nodes at every level. Use nextEqual pointer to link the nodes with the same year together. Use pillar to store the height of the pillar.

My EventList class:

```
class EventList {
    Random randseq;
    public static Event head,tail;
    public static int highest; //keep track the highest pillar
    int randomHeight()
    {
    int v = 1;
    while (randseq.nextBoolean()) { v++; }
    return v;
    }

    public EventList()
    {
    randseq = new Random(58243); // You may seed the PRNG however
you like.
    head = new Event(Integer.MIN_VALUE,"-oo");
    tail = new Event(Integer.MAX_VALUE,"+oo");
```

```
    head.pillar = 1000;
    tail.pillar = 1000;
    head.next = new Event[1000];
    tail.prev = new Event[1000];
    for(int i = 0; i < 1000; i++){
        head.next[i] = tail;
        tail.prev[i] = head;
    }
    highest = 0;//Initialize head and tail,highest
    }
```

Explain:Initialize head and tail, they have 2 arrays of 1000 length,point to each other, no node in the list at first, highest = 0.

```
    @Override
    public String toString() {
        String total = "";
        for(Event i = head.next[0]; i != tail; i = i.next[0])
            for(Event j = i; j!=null; j = j.nextEqual)
                total+=j.toString()+"\n";
        //use a chain to store the events with the same year,
        //the end of the chain point to null
        return total;
    }
```

Explain:print out every node including the nodes on the chain.

```
    public void insertSame(Event e,Event base){
        if(base.nextEqual==null){
            base.nextEqual = e;
        }//if it's the first element except the base, add it to the end of the chain
        //and it's nextEqual is null
        else{
            e.nextEqual = base.nextEqual;
            base.nextEqual = e;
        }//if it's not the 1st element, add it to the start of the chain
        //between the base and the first
    }
```

Explain:Chain insert. Input is the base Event and the target Event, chain it between the base and the first one.

```
    public void insert(Event e)
```

```
    {
        Event equal = find(e.year);//use year to find,if find result
is not null
        if(equal!=null){
            insertSame(e,equal);//chain it to the base, and end
        }
        else{
        int t = randomHeight(); //pillar height
        e.pillar = t;
        e.next = new Event[t];
        e.prev = new Event[t];  //construct 2 pillars for the new
node

        int l = Math.max(0, highest-1);//if highest=0(no node in the
list,l=0
        //if highest>0, l= highest-1
        Event x = head; //start from the head
        Event y;
        while(l>=0){
            y = x.next[l];
            if(y.year<e.year)//y is not big enough,move x=y
                x = y;
            else{
                if(l < e.pillar){  //y is too big,l--,and link
properly
                    x.next[l] = e;
                    e.prev[l] = x;
                    e.next[l] = y;
                    y.prev[l] = e;
                    }
                l--;
                }
        }
        if(e.pillar>highest){ //exceed the original highest, link
head and tail at this part
            for(int i = highest; i < e.pillar;i++){
                head.next[i] = e;
                e.prev[i] = head;
                e.next[i] = tail;
                tail.prev[i] = e;
            }
            highest = e.pillar;
        }
        }
```

```
        }
```

Explain:First,find if there is a base is the list, if does, insertSame.If not, find a proper place to put it, always start from the left,need to consider when highest = 0, if highest!=0, l=highest-1, start from this level.

```
    public Event find(int year){
        if(highest==0)//empty list, return null
            return null;
        int l = highest - 1;
        Event x = head;//start from the head
        Event y;
        while(l>=0){
            y = x.next[l];
            if(y.year == year)//find, return
                return y;
            else
                if(y.year<year)//y is not big enough, move x to y
                    x=y;
                else//y is too big, l--
                    l--;
            }
        return null;//l=0, and can't find it, return null
    }
```

Explain:Return the event or null.

```
    public Event findApproximate(int year){
        if(highest==0)
            return null;//empty return null
        else{
        int l = highest - 1;
        Event x = head;//start from the head
        Event y;
        while(l>=0){
            y = x.next[l];
            if(y.year == year)
                return y;//y.year=year
            else
                if(y.year<year)
                    x=y;
                else
                    l--;
            }
            return x;//can't find the exact one,return the prev(year)
```

```
        }//the return.year is always<=year
    }
```

Explain:The event we return is the approximate of the year, it's <=year: the exact one if it exists or the biggest one smaller than the year.

```
public Event findBiggerApproximate(int year){
        if(highest==0)
            return null;//empty return null
        else{
        int l = highest - 1;
        Event x = head;//start from the head
        Event y;
        while(l>=0){
            y = x.next[l];
            if(y.year == year)
                return y;//y.year=year
            else
                if(y.year<year)
                    x=y;
                else
                    l--;
            }
            return x.next[0];//can't find the exact one,return the succ(year)
        }//the return.year is always>=year
    }
```

Explain:similar with the last one, but return.year>=year

```
    public void removeChain(Event x){
        Event y,z;
        for(int i = 0; i<x.pillar;i++){
            y = x.prev[i];
            z = x.next[i];
            y.next[i] = z;
            z.prev[i] = y;
        }//unlink x level by level
    }

    public void remove(int year)
    {
        Event x = find(year);//find it
        if(x!=null)
            removeChain(x);
    }
```

```
    public Event [] returnChain(Event x){
        List<Event> returnlist = new ArrayList<Event>();
        for(Event i = x; i !=null; i = i.nextEqual)
            returnlist.add(i);
        Event[] chain = new Event[returnlist.size()];
        returnlist.toArray(chain);//use arraylist to store the
chain
        return chain;
    }
```

Explain:Unlink the base at every level.

```
    public Event [] findMostRecent(int year)
    {
        //year is no less than the smallest in the list
        if(year>=head.next[0].year){
            Event x = findApproximate(year);//x.year<=year
            return returnChain(x);
        }
        else//return null, because too small
            return null;
    }
```

Explain:Use findApproximate to find it, x.year<=year

```
    public Event [] findRange(int first, int last)
    {
        List<Event> rangelist = new ArrayList<Event>();
        if(first>tail.prev[0].year||last<head.next[0].year)
            return null;//first is bigger than the biggest, or last
is smaller than the smallest
        else{
            Event fevent =
findBiggerApproximate(first);//fevent<=first
            Event levent = findApproximate(last);//levent<=last
            if(fevent==levent.next[0])
                return null;//list has 1699,1700,1702,want to find
from 1701 to 1701
            //fevent=1702, levent=1700, so null
Find (1700,1701),fevent=1700,levent=1700,have one
            for(Event i = fevent; i!=levent.next[0];
i=i.next[0])//from fevent to levent,
                //and fevent!=levent
                for(Event j = i; j !=null; j = j.nextEqual)//add the
chain
```

```
                rangelist.add(j);
            Event[] range = new Event[rangelist.size()];
            rangelist.toArray(range);
            return range;
            }
    }
}
```

Explain:Use findApproximate to find it, x.year<=year, if fevent and levent are equal to each other,and it's not in the list,return null. For example,1699,1700,1702,find(1701,1701),fevent=1702,levent=1700, return null.But find(1700,1700),fevent=levent=1700,should return 1700.

Part 2:

```
public void doubleHeadTailHeight(int t){
       int h;
       for(h =2*head.pillar; h<t;h=2*h);
//double h until h>=t.
       Event[] headTemp = new Event[head.pillar];
       Event[] tailTemp = new Event[tail.pillar];
       for(int i = 0; i < head.pillar; i++){
           headTemp[i] = head.next[i];
           tailTemp[i] = tail.prev[i];
       }
//copy to the temp array, only copy head.next and tail.prev
       head.prev = new Event[h];
       head.next = new Event[h];
       tail.next = new Event[h];
       tail.prev = new Event[h];
//build new size arrays
       for(int i = 0; i<head.pillar;i++){
           head.next[i] = headTemp[i];
           tail.prev[i] = tailTemp[i];
       }//don't need to change other links because the pointers to
head and tail don't change
       for(int i = head.pillar; i <h; i++){
           head.next[i]=tail;
           tail.prev[i]=head;
       }//link the other together
       head.pillar = h;
       tail.pillar = h;
    }
```

Explain:when insert, use if(t>head.pillar) to decide whether or not to increase the height of head and tail's pillar.

```
if(t>head.pillar)
    doubleHeadTailHeight(t);
```

Use single-linked:

I use max to represent the max node in the list, and remove all the prev, since I used findApproximate(year) to return something no bigger than the year, and findBiggerApproximate(year) to return something no less than the year. So my findrange and findmostresent don't change. My change is my remove, I find the target, and start from its top level, unlink it level by level.

```
 public void remove(int year)
    {
        Event target = find(year);//find year.
        if(target!=null){
        int l = target.pillar;//start from l level
        Event x = head;
        Event y;
        while(l>=0){
            y = x.next[l];
            if(y.year<year)
                x = y;
            else
                if(y.year==year)//if we find y,remove y from level l
                    x.next[l]=y.next[l];
                else
                    l--;
        }
        }
    }
```

Explain:This is my remove, in part 1, I used removeChain to remove things.

My findBiggerApproximate(year) and findApproximate(year) remains the same, so are my findrange and findmostresent.

Other functions just removed the prev part. I overwritten all the codes in my EventList.java.