# LANČANE LISTE

# Zamena mesta susedima u listi

```
template <class T>
void SLList<T>::SwapNeighbors(SLLNode<T>* ptr,
                              SLLNode<T>* prev) {
    if (prev == NULL) {
        head = ptr->next;
        ptr->next = ptr->next->next;
        head->next = ptr;
    } else {
        prev->next = ptr->next;
        ptr->next = ptr->next->next;
        prev->next->next = ptr;
    }
}
```

# Zamena mesta za dva elementa u listi

```
template <class T>
void SLList<T>::Swap(SLLNode<T>* ptr1, SLLNode<T>* prev1, SLLNode<T>*
   ptr2, SLLNode<T>* prev2) {
    if (ptr1 == ptr2) {
        return;
    } else if (ptr1->next == ptr2) {
        SwapNeighbors(ptr1, prev1);
    } else if (ptr2->next == ptr1) {
        SwapNeighbors(ptr2, prev2);
    } else {
        if (prev1 == NULL)
            head = ptr2;
        else
            prev1->next = ptr2;
        if (prev2 == NULL)
            head = ptr1;
        else
            prev2->next = ptr1;
        SLLNode<T>* tmp = ptr1->next;
        ptr1->next = ptr2->next;
        ptr2->next = tmp;
    }
}
```

# Bubble Sort za lančanoj listi

```
template <class T>
void SLList<T>::BubbleSort() {
    SLLNode<T> *ptr1;
    SLLNode<T> *ptr2, *prev2;
    for (ptr1=tail; ptr1!=head; ptr1=prev2) {
        prev2 = NULL;
        for (ptr2=head; ptr2->next!=ptr1; ) {
            if (ptr2->info > ptr2->next->info) {
                SwapNeighbors(ptr2, prev2);
                if (prev2 != NULL) {
                    prev2 = prev2->next;
                } else {
                    prev2 = head;
                }
            } else {
                prev2 = ptr2;
                ptr2=ptr2->next;
            }
        }
```

```
            if (ptr2->info > ptr2->next->info) {
                SwapNeighbors(ptr2, prev2);
                if (prev2 != NULL) {
                    prev2 = prev2->next;
                } else {
                    prev2 = head;
                }
            } else {
                prev2 = ptr2;
            }
        }
    }
}
```

# Selection Sort za lančanu listu

```
template <class T>
void SLList<T>::SelectionSort() {
    SLLNode<T> *ptr, *prev;
    SLLNode<T> *ptr1, *prev1, *ptr2, *prev2;
    prev1 = NULL;
    for (ptr1=head; ptr1!=NULL; ptr1=ptr1->next) {
       ptr = ptr1;
       prev = prev1;
       prev2 = ptr1;
       for (ptr2=ptr1->next; ptr2!=NULL; ptr2=ptr2->next) {
            if (ptr->info > ptr2->info) {
                ptr = ptr2;
                prev = prev2;
            }
            prev2 = ptr2;
       }
       if (ptr1 != ptr) {
           Swap(ptr1, prev1, ptr, prev);
            ptr1 = ptr;
       }
       prev1 = ptr1;
    }
}
```

# Insertion Sort za lančanu listu

```
template <class T>
void SLList<T>::InsertionSort() {
    SLLNode<T> *ptr1, *prev1, *ptr2, *prev2;
    prev1 = head;
    for (ptr1=head->next; ptr1!=NULL; ) {
        prev2 = NULL;
        ptr2 = head;
        while (ptr2!=ptr1 && ptr1->info<ptr2->info) {
            prev2 = ptr2;
            ptr2 = ptr2->next;
        }
        if (ptr1 != ptr2) {
            if (prev2 != NULL) {
                prev1->next = ptr1->next;
                prev2->next = ptr1;
                ptr1->next = ptr2;
            } else {
                prev1->next = ptr1->next;
                head = ptr1;
                ptr1->next = ptr2;
            }
            ptr1 = prev1->next;
        } else {
            prev1 = ptr1;
            ptr1 = ptr1->next;
        }
    }
}
```

# Umetanje u sortiranu listu

```
template <class T>
void SLList<T>::InsertInSorted(SLLNode<T> *ptr) {
   SLLNode<T> *ptr1, *prev1;
   prev1 = NULL;
   ptr1 = head;
   while (ptr1!=NULL && ptr->info<ptr1->info) {
        prev1 = ptr1;
        ptr1 = ptr1->next;
   }
   if (prev1 != NULL) {
        prev1->next = ptr;
        ptr->next = ptr1;
   } else {
        head = ptr;
        ptr->next = ptr1;
   }
}
```

# Bucket Sort

- Pretpostavke:
  - Vrednost elementa niza je u opsegu [0, 1)
  - Elemenata su uniformno raspoređeni
- Složenost algoritma je linearna O(n)
- Zahteva dodatni memorijski prostor veličine n
- Ideja:
  - Podeliti niz na N opsega
  - Sekvencijalno dodavati elemente niza u odgovarajuće bucket-e, tako da svaki bucket bude uređen
  - Spojiti buckete u uređeni niz

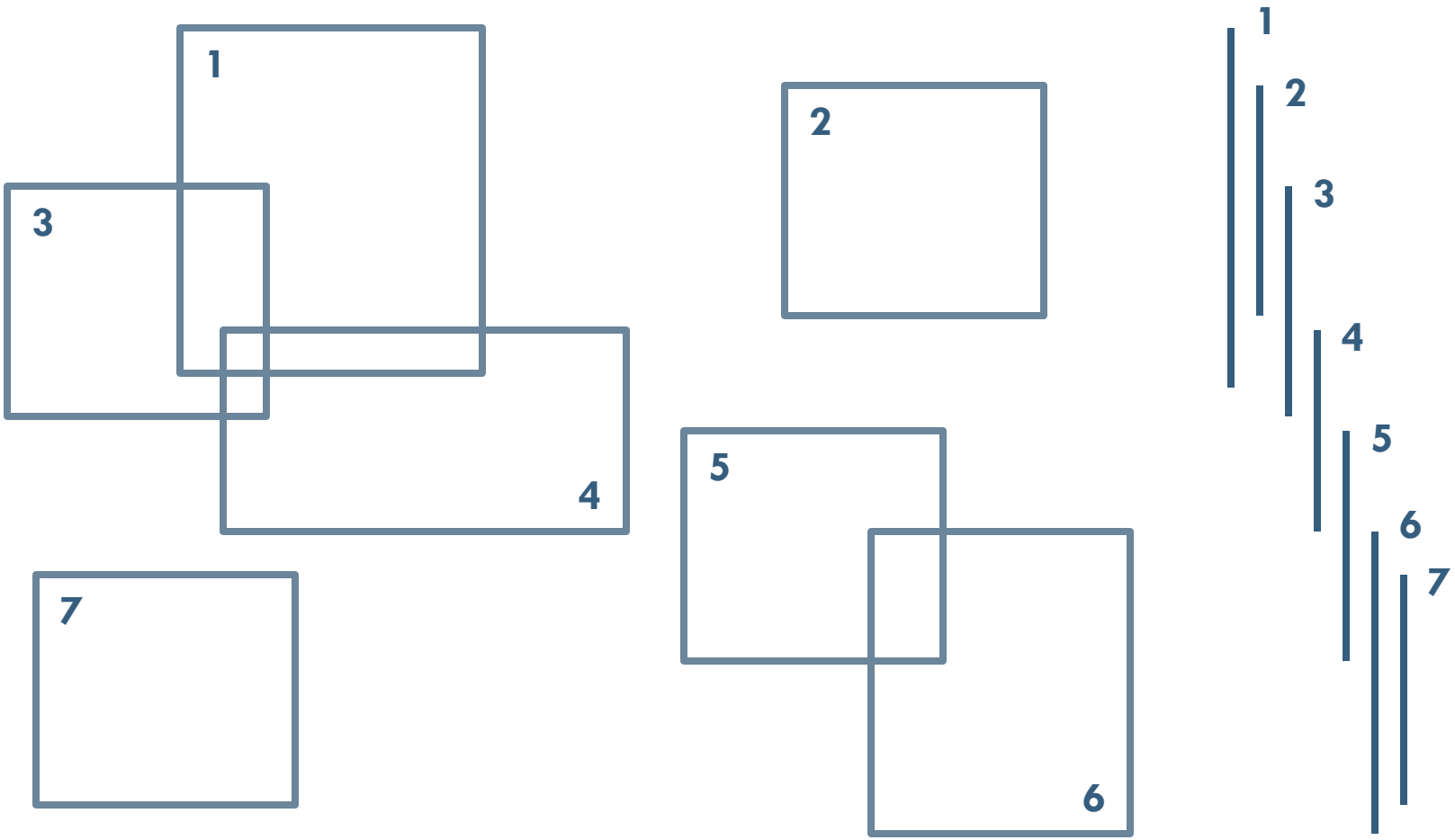
# Bucket Sort

```
void BucketSort(double a[], double b[], int n) {
   SLLNode<double> *pCvor;
   SLList<double> *lista = new SLList<double>[n];
   int i;
   for (i=0; i<n; i++) {
        pCvor = new SLLNode<double>(a[i]);
        lista[(int)(n*a[i])].InsertInSorted(pCvor);
   }
   int ind = 0;
   for (i=0; i<n; i++) {
        while (!lista[i].isEmpty()) {
               b[ind++] = lista[i].deleteFromHead();
        }
   }
   delete[] lista;
}
```

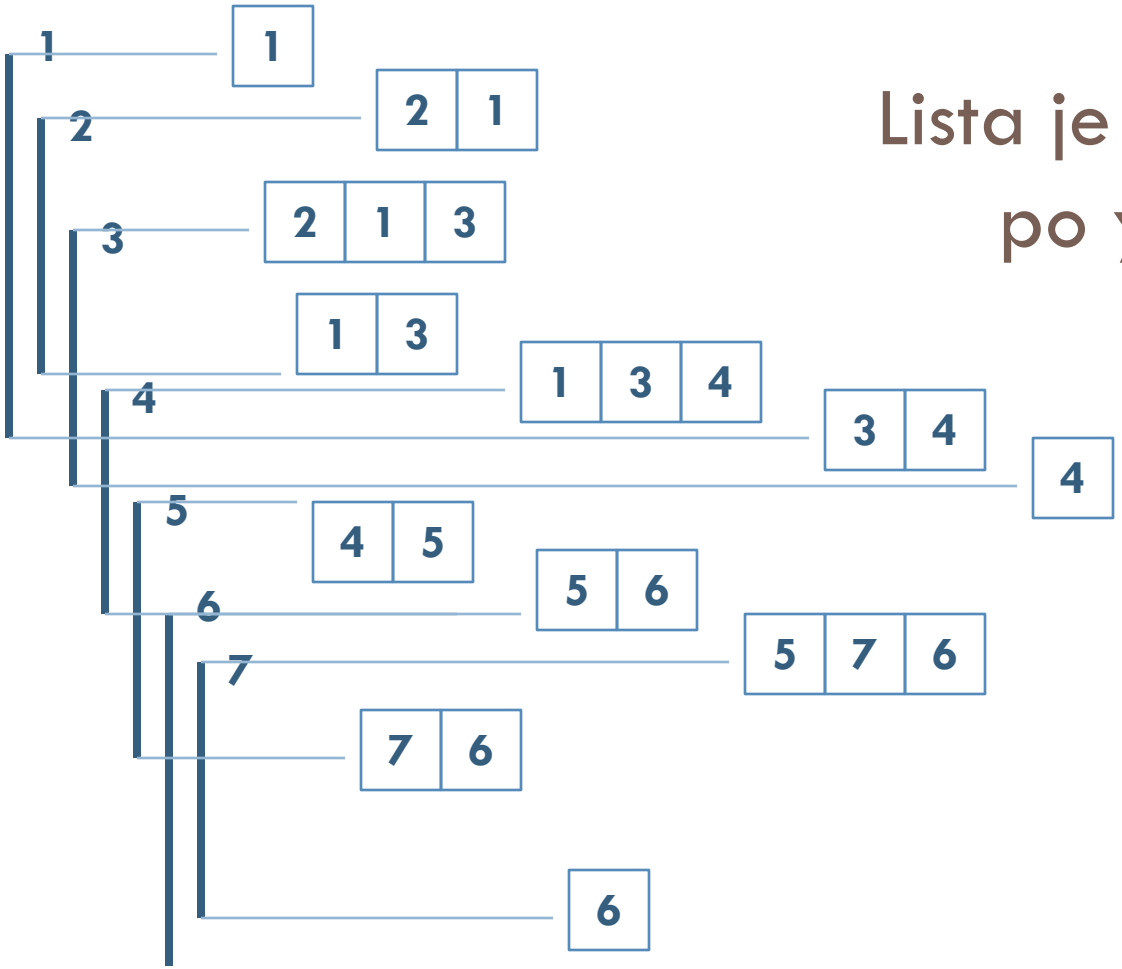
# Presek uređenih pravougaonika

- Pravougaonici imaju ivice paralalene x i y osi i pamte se kao niz (xmin, ymin, xmax, ymax)
- Ulaz: Niz pravougaonika uređenih po minimalnoj y koordinati
- Izlaz: Niz parova pravougaonika koji se seku
- Trivijalno rešenje: uporediti svaki sa svakim
- Optimizicaja: iskoristiti uređenost ulaznog niza da bi se suzio skup za poređenje

# Presek uređenih pravougaonika



# Presek uređenih pravougaonika




Lista je uređena po ymax

# Presek uređenih pravougaonika

```
class RectXY {
public:
   int xmin;
   int ymin;
   int xmax;
   int ymax;

   RectXY();
   RectXY(int x1, int x2, int y1, int y2);
   RectXY(const RectXY& rc);

   bool operator < (const RectXY& rc);
   bool operator > (const RectXY& rc);
   const RectXY& operator = (const RectXY& rc);

   void Print();
};
```

# Presek uređenih pravougaonika

```
bool RectXY::operator <  (const RectXY& rc) {
  return ymax < rc.ymax;
}

bool RectXY::operator > (const RectXY& rc) {
  return ymax > rc.ymax;
}

void RectXY::Print() {
  cout << "( " << xmin << " " << ymin << " " <<
  xmax << " " << ymax << " )   " ;
}
```

# Presek uređenih pravougaonika

```
void FindIntercted(SLList<RectXY>& llist, RectXY& rc) {
   RectXY rcList;
   SLLNode<RectXY> *ptr;
   ptr = llist.head;
   while (ptr != NULL) {
        rcList = ptr->info;
        if (rc.xmin <= rcList.xmax
                        && rc.xmax >= rcList.xmin) {
              rc.Print();
              rcList.Print();
              cout << endl;
        }
        ptr = ptr->next;
   }
}
```

# Presek uređenih pravougaonika

```
void IntersectRect(RectXY arRect[], int n) {
   RectXY rc, rcList;
   SLList<RectXY> llist;
   for (int i=0; i<n; i++) {
        bool bStop = false;
        while (!bStop && !llist.isEmpty()) {
               rcList = llist.getHeadEl();
               if (arRect[i].ymin > rcList.ymax)
                      llist.deleteFromHead();
               else
                      bStop = true;
        }
        FindIntercted(llist, arRect[i]);
        SLLNode<RectXY> *pNode = new SLLNode<RectXY>(arRect[i]);
        llist.InsertInSorted(pNode);
   }
}
```