

Vistula, IT Faculty, 2014

Algorithms and Complexity

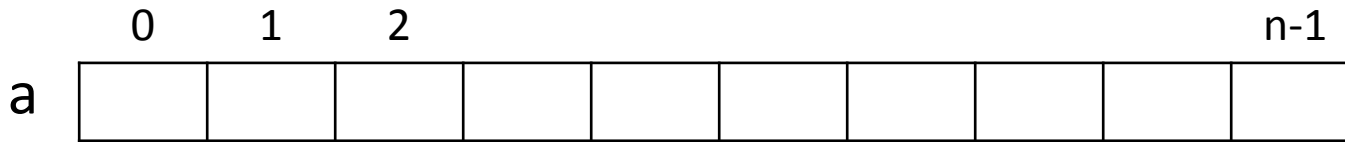
Dmitry A. Zaitsev

<http://daze.ho.ua>

Lecture 9:

Sorting, merging, and string algorithms

Preliminaries of sorting



A partial order relation \leq

$$a[i] \leq a[i+1], 0 \leq i < n-1$$

Simplest sorting:

Bubble

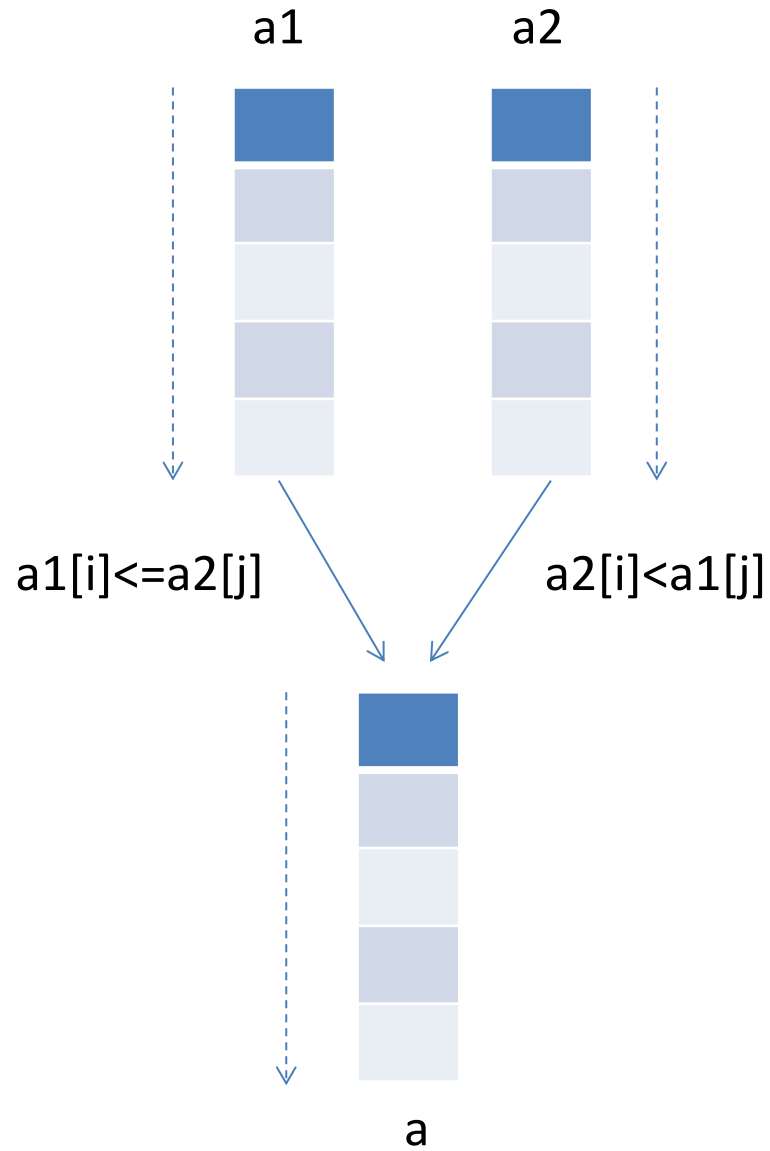
Sequential minimums

Insertion

Sorting by comparison of elements not better than

$$n \cdot \log n$$

Merging two arrays (lists)



Merging two arrays (lists)

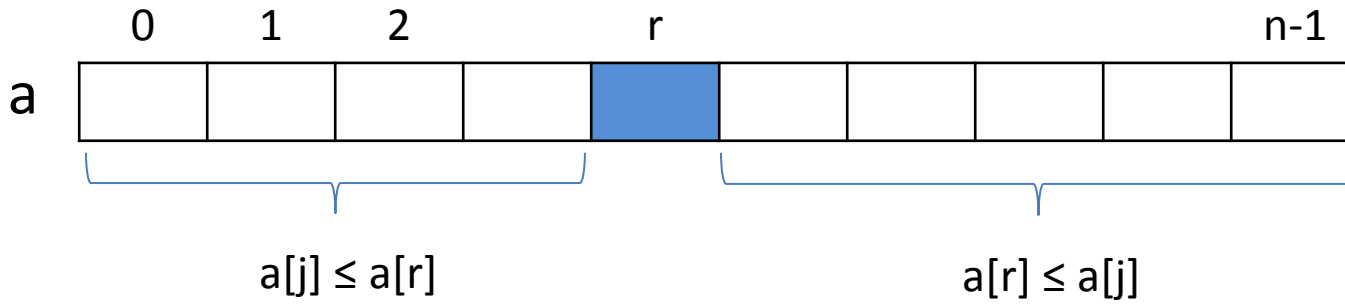
```
struct elist1 * merge(struct elist1 * a1, struct elist1 * a2)
{
    struct elist1 *x1=a1, *x2=a2, *h=NULL, *c=NULL;
    while(x1!=NULL || x2!=NULL)
    {
        if(x2==NULL || (x1->cont < x2->cont))
        {
            if(h==NULL)h=x1;
            if(c!=NULL) c->next=x1;
            c=x1; x1=x1->next;
        }
        else
        {
            if(h==NULL)h=x2;
            if(c!=NULL) c->next=x2;
            c=x2; x2=x2->next;
        }
    }
    if(c!=NULL) c->next=NULL;
    return(h);
}
```

Sorting via merging

```
struct elist1 * mergesort(struct elist1 * a)
{
    struct elist1 * a2;
    if(llen(a)<2) return(a);
    else
    {
        a2=ldiv2(a);
        return( merge(mergesort(a), mergesort(a2)) );
    }
}

struct elist1 * ldiv2(struct elist1 * a)
{
    int n = llen(a), n1 = n/2, n2 = n-n1;
    struct elist1 *x, *c=a;
    while(n1-->0) c=c->next;
    x=c->next; c->next=NULL;
    return(x);
}
```

Quick sort



1. Choose $a[r]$
2. Put less or equal elements to the left and greater or equal elements to the right of $a[r]$
3. Repeat the same with the parts

Specific methods of sorting integers and strings

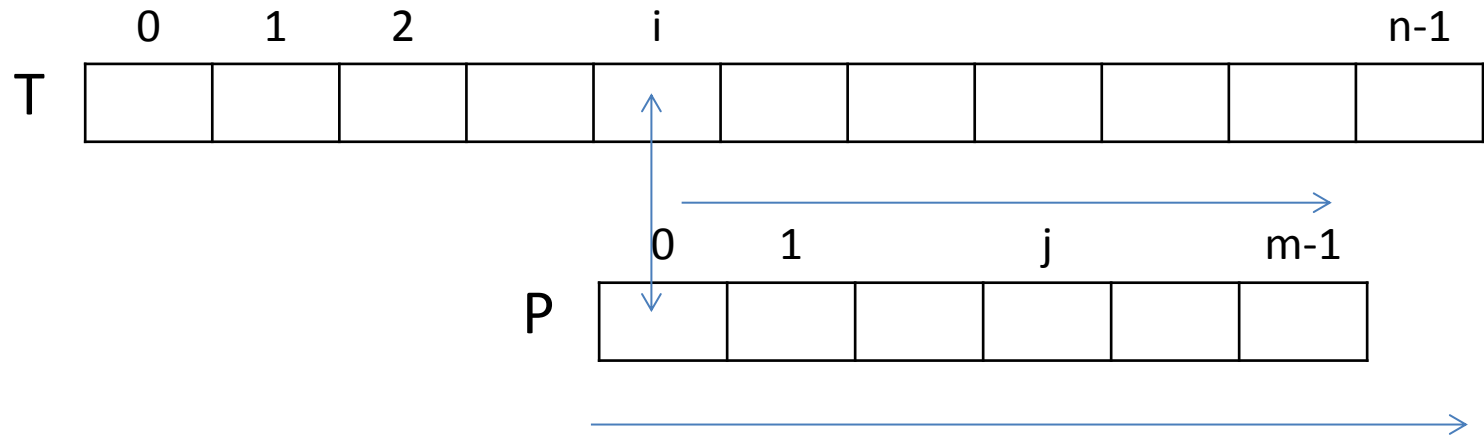
Radix sorting: $0 \leq a[i] \leq m-1$

1. Initialize m empty queues (buckets) for each of integers from 0 to $m-1$
2. Scan array a from left to right and place $a[i]$ into $q[a[i]]$
3. Concatenate queues

$O(n)$

String sorting

String algorithms: substring matching



String T , pattern P

```
for(i=0; i<n-m;i++)  
{  
    found=1;  
    for(j=0; j<m;j++)  
        if( T[i+j]!=P[j]) {found=0;break;}  
    if(found) printf("%d ", i);  
}
```

$O(n \cdot n)$

The Knuth-Morris-Pratt (KMP) algorithm

```
q=0; i=0;
while (i < n) // P[0,...,q-1] == T[i - q,...,i-1]
{
    if (P[q] == T[i])
    {
        q++; i++;
        if (q == m)
        {
            printf("", i-q);
            q=offs(q); //slide the pattern to the right
        }
    }
    else // a mismatch occurred
        if (q == 0) i++; else q=offs(q);
}
```

$O(n+n)$