Αλγόριθμοι Divide-and-Conquer

# Περίληψη

- Αλγόριθμοι Divide-and-Conquer
- Master Theorem
- Παραδείγματα
  - Αναζήτηση
  - Ταξινόμηση
  - Πλησιέστερα σημεία
  - Convex Hull

# Αλγόριθμοι Divide-and-Conquer

- Γενική Μεθοδολογία
  - Το πρόβλημα διαιρείται (divide) σε δύο ή περισσότερα υπο-προβλήματα
    - Είναι προτιμότερο τα υπο-προβλήματα να έχουν περίπου το ίδιο μέγεθος
  - Το κάθε υπο-πρόβλημα λύνεται ξεχωριστά (conquer)
    - Πολλές φορές το πρόβλημα λύνεται αναδρομικά (recursively)
    - Ειδικές λύσεις μπορεί να δοθούν για πολύ μικρά προβλήματα
  - Εάν χρειάζεται, οι λύσεις των υπο-προβλημάτων συνδυάζονται για να βρεθεί η λύση στο αρχικό πρόβλημα (conquer)

# Προβλήματα Αναζήτησης

- Είσοδος: Λίστα $A[0,\ldots,n-1]$, Κλειδί $K$
- Έξοδος: Η θέση του κλειδιού στη λίστα (εάν υπάρχει)

```
Srch(A[k…l], K)
  if(k = l)
      if A[k]= K return k;
      else return -1;
  m=floor ((l+k)/2);
  return max(Srch(A[k…m],K), Srch(A[m+1…l],K));
```

- Είναι ο Srch() καλύτερος από τον Brute-Force Search();
- Divide-and-conquer δεν εγγυάται πιο αποδοτικούς αλγόριθμους!

# Αναδρομική Σχέση Αλγορίθμων Divide-and-Conquer

- Ένα πρόβλημα μεγέθους $n$ διαιρείται σε $b>1$ υπο-προβλήματα μεγέθους $n/b$.
- Από τα $b>1$ υπο-προβλήματα τα $a>0$ θα πρέπει να λυθούν.
- Για τη διαίρεση (και επανασύνδεση) ενός προβλήματος $n$ σε $b$ υπο-προβλήματα χρειάζεται χρόνος $f(n)$
- Η αναδρομική σχέση για το χρόνο που χρειάζεται για την ολοκλήρωση του αλγορίθμου $T(n)$ δίνεται από

# Master Theorem

- Δεδομένης της αναδρομικής σχέσης

$$T(n) = aT\left(n/b\right) + f(n)$$

Εάν ισχύει $f(n) \in \Theta(n^d)$, όπου $d \geq 0$, τότε

# Προβλήματα Αναζήτησης

```
Srch(A[k…l], K)
   if(k = l)
       if A[k]= K return k;
       else return -1;
   m=floor ((l+k)/2);
   return max(Srch(A[k…m],K), Srch(A[m+1…l],K));
```

- Το πρόβλημα διαιρείται σε $b=2$ υπο-προβλήματα.
- Θα πρέπει να λυθούν και τα δύο υπο-προβλήματα $a = 2$.
- Ο χρόνος διαίρεσης είναι σταθερός $f(n)= c$.

# Προβλήματα Ταξινόμησης

- Είσοδος: Λίστα $A[0,\ldots,n-1]$
- Έξοδος: Ταξινομημένη λίστα $A[0,\ldots,n-1]$

```
mergesort(A[0,…,n-1])
  if n=1 return;
  m= floor(n/2)-1;
  B[]= A[0,…,m];
  C[]= A[m+1,…,n-1];
  mergesort(B);
  mergesort(C);
  merge(B,C,A);
```

# Merge: Επανασύνδεση δύο ταξινομημένων λιστών

- Πώς επανασυνδέουμε δύο ταξινομημένες λίστες έτσι που η λίστα που θα προκύψει να παραμείνει ταξινομημένη

```
merge(B[0…p-1],C[0…q-1],A[])
  i=0; j=0; k=0;
  while i<p and j<q
     if B[i] <= C[j]
          A[k]= B[i]; i= i+1;
     else A[k]= C[j]; j= j+1;
     k=k+1;
  if i=p copy C[j,…,q-1] to A[k,…,p+q-1];
  else copy B[i,p-1] to A[k,…,p+q-1];
```

# Απόδοση της Merge() και MergeSort()

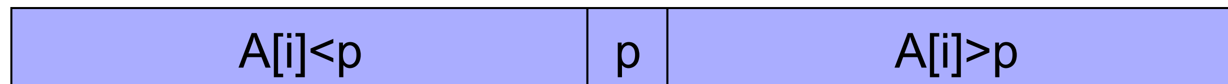
```
merge(B[0…p-1],C[0…q-1],A[])
  i=0; j=0; k=0;
  while i<p and j<q
      if B[i] <= C[j]
           A[k]= B[i]; i= i+1;
      else A[k]= C[j]; j= j+1;
      k=k+1;
  if i=p copy C[j,…,q-1] to A[k,…,p+q-1];
  else copy B[I,p-1] to A[k,…,p+q-1];
```

# Παράδειγμα MergeSort()

```
mergesort(A[0,…,n-1])
   m= floor(n/2)-1;
   B[]= A[0,…,m];
   C[]= A[m+1,…,n-1];
   mergesort(B);
   mergesort(C);
   merge(B,C,A);
```

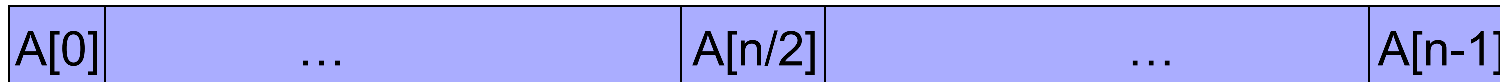
# QuickSort()

- Ακόμα ένα παράδειγμα αλγόριθμου ταξινόμησης μιας λίστας που είναι βασισμένος σε divide-and-conquer
- Βασική Ιδέα:
  - Για κάθε στοιχείο της λίστας προσπαθούμε να βρούμε σε πιο ακριβώς σημείο της λίστας θα πρέπει να τοποθετηθεί



# Προβλήματα Αναζήτησης σε **Ταξινομημένη** Λίστα

- Είσοδος: Ταξινομημένη Λίστα $A[0,\ldots,n\text{-}1]$, Κλειδί $K$
- Έξοδος: Η θέση του κλειδιού στη λίστα (εάν υπάρχει)




- Η διαδικασία αυτή συνεχίζεται με το ίδιο τρόπο για τη πιο μικρή λίστα.

# Προβλήματα Αναζήτησης σε Ταξινομημένη Λίστα

- Είσοδος: Ταξινομημένη Λίστα $A[0,\ldots,n-1]$, Κλειδί $K$
- Έξοδος: Η θέση του κλειδιού στη λίστα (εάν υπάρχει)

```
BinSearch(A[],b,e,K)
  m= floor((e+b)/2);
  if A[m]=K return m;
  if A[m] > K return BinSearch(A, b, m-1, K);
  return BinSearch(A, m+1, e, K);
```

- Είναι ορθός ο πιο πάνω αλγόριθμος ή υπάρχουν περιπτώσεις στις οποίες αποτυγχάνει;
  - Αποτυγχάνει όταν το K δεν είναι στη λίστα!

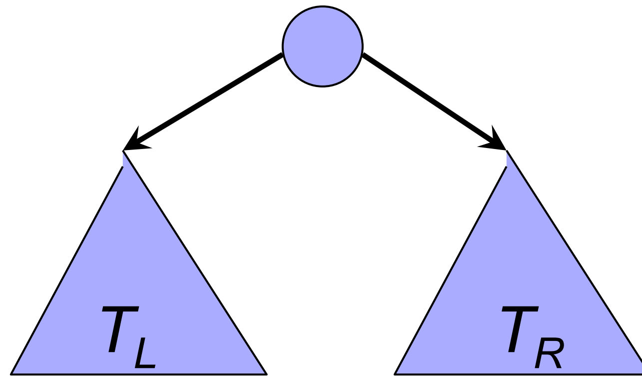
# Απόδοση Αλγορίθμου

```
BinSearch(A[],b,e,K)
   m= floor((e+b)/2);
   if A[m]=K return m;
   if A[m] > K return BinSearch(A, b, m-1, K);
   return BinSearch(A, m+1, e, K);
```

- Αριθμός συγκρίσεων

# Προβλήματα «Εξερεύνησης» Δυαδικών Δέντρων (Tree Traversal)

- Σε ένα δυαδικό δέντρο, ένας κόμβος μπορεί να θεωρηθεί σαν μια ρίζα τα παιδία της οποίας αποτελούν δύο ανεξάρτητα δέντρα.




- Για την καλύτερη «οργάνωση» της εξερεύνησης, κάποιος μπορεί να εξερευνήσει πρώτα το ένα από το υπο-δέντρα και στην συνέχεια το άλλο
- Τρεις τρόποι εξερεύνησης: **preorder**, **inorder**, **postorder**

# Πολλαπλασιασμός Ακεραίων Αριθμών

- Πολλαπλασιάστε δύο διψήφιους αριθμούς
  - A=ab, D=cd
  - A*D=;

$$A = a10^1 + b10^0 \qquad D = c10^1 + d10^0$$

$$A * D = (a*c)10^2 + (c*b + a*d)10^1 + (b*d)10^0$$

- Εάν $n$ είναι ο αριθμός ψηφίων του κάθε αριθμού, τότε χρειαζόμαστε $n^2$ πολλαπλασιασμούς!
- Μπορείτε να βρείτε πιο αποδοτικό αλγόριθμο;

# Πολλαπλασιασμός Ακεραίων Αριθμών

- Πολλαπλασιάστε δύο αριθμούς με $n$ ψηφία ο καθένας (υποθέστε πως το $n$ είναι ζυγό).

$$A = \sum_{i=0}^{n-1} a_i 10^i \qquad B = \sum_{i=0}^{n-1} b_i 10^i$$

$$A * B = (a_h * b_h)10^n + (a_h * b_l + a_l * b_h)10^{n/2} + (a_l * b_l)10^0$$

- Η πιο πάνω σχέση μπορεί να χρησιμοποιηθεί αναδρομικά έτσι ώστε να υπολογιστεί το γινόμενο δύο μεγάλων αριθμών πιο αποδοτικά.
- Ένας τέτοιος αλγόριθμος μπορεί να είναι πολύ χρήσιμος στην κρυπτογραφία
  - Πολλαπλασιασμός μεγάλων «κλειδιών»
- Παρόμοιος αλγόριθμος ισχύει και στην περίπτωση πολλαπλασιασμού πινάκων

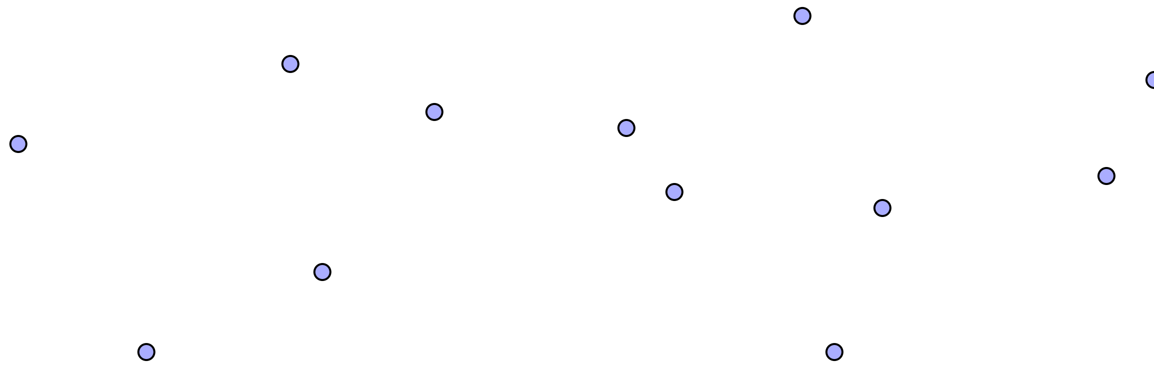
# Πολλαπλασιασμός Ακεραίων

- Είσοδος: Οι αριθμοί $A[0,\ldots,n-1]$, και $B[0,\ldots,n-1]$
- Έξοδος: $A*B$

```
multiply(A[],B[])%n is a power of 2
  n=sizeof(A[]); m= n/2;
  if(n=1) return A[0]*B[0];
  Al= A[0…m]; Ah=A[m+1…s];
  Bl= B[0…m]; Bh=B[m+1…s];
  AA= multiply(Ah, Bh);
  BB= multiply(Al, Bl);
  CC= multiply(Al+Ah, Bl+Bh)-AA –BB;
  return AA*(10^n)+ CC*(10^n/2) +BB;
```

# Πρόβλημα εύρεσης των δύο πλησιέστερων σημείων

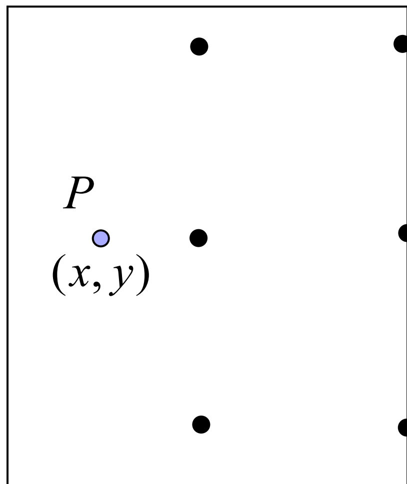
- Δεδομένης μιας λίστας με $n$ σημεία $\{P_0,\ldots,P_{n-1}\}$ βρείτε ποια δύο σημεία έχουν την μικρότερη απόσταση.
- Υποθέστε πως η λίστα $\{P_0,\ldots,P_{n-1}\}$ είναι ταξινομημένη με βάση τη συντεταγμένη $x$.



- Χωρίστε τα σημεία σε δύο «μισές» λίστες
- Λύστε τα δύο υπο-προβλήματα
- «Συνδυάστε» τα δύο αποτελέσματα

# Πρόβλημα εύρεσης των δύο πλησιέστερων σημείων

- Επανασύνδεση των αποτελεσμάτων



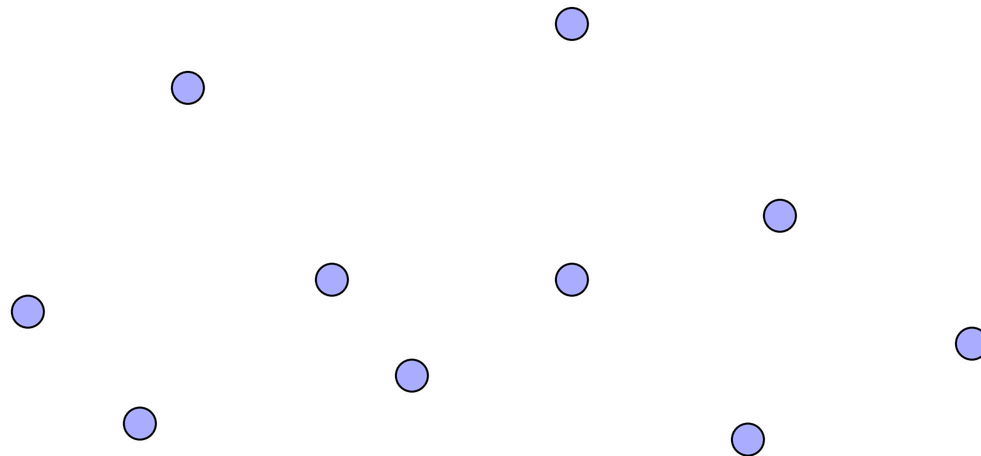

- Για *κάθε* σημείο που βρίσκεται στα αριστερά της διαχωριστικής γραμμής $x=c$ πρέπει να ελέγξουμε εάν υπάρχει σημείο σε απόσταση μικρότερη από $d$.
- Πόσα τέτοια σημεία μπορεί να υπάρχουν;
  - 

- Απόδοση Αλγορίθμου:

- Πως αλλάζει η απόδοση εάν η λίστα **δεν** είναι ταξινομημένη;

# Πρόβλημα Εύρεσης Convex Hull

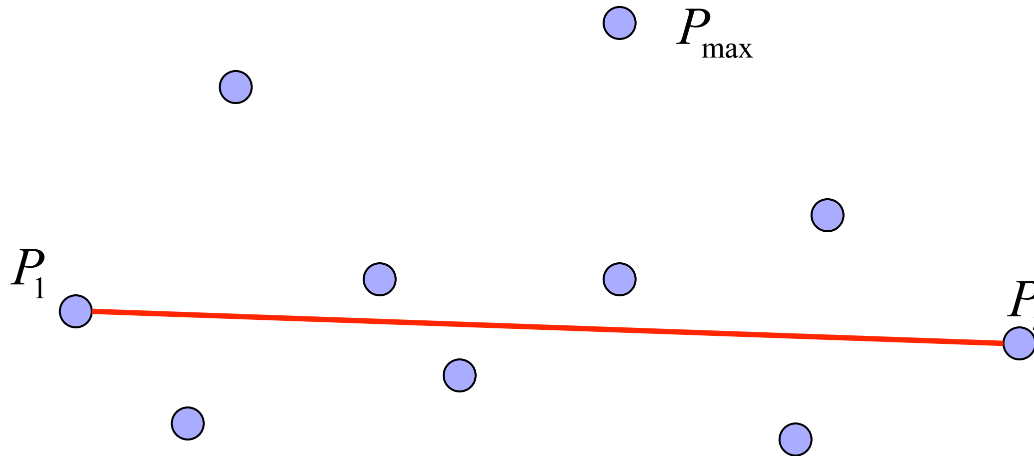
- Δεδομένης μιας λίστας με $n$ σημεία $\{P_1,\ldots,P_n\}$ βρείτε το **μικρότερο** κυρτό χώρο που εμπερικλείει όλα τα σημεία $\{P_1,\ldots,P_n\}$ (**convex hull**).
- Θεωρείστε πως τα σημεία είναι ταξινομημένα με βάση τη συντεταγμένη $x$.



- Τα σημεία $P_1$ και $P_n$ είναι κορυφές του convex hull!
- Το $P_{\max}$ είναι επίσης σημείο του convex hull.
- Τα σημεία μέσα στο τρίγωνο $P_1P_nP_{\max}$ δεν είναι κορυφές του convex hull.
    - Τα σημεία αυτά είναι δεξιά της $P_1P_{\max}$ και δεξιά της $P_{\max}P_n$!

# Πρόβλημα Εύρεσης Convex Hull

- Ποιο σημείο είναι πιο μακριά από την ευθεία $P_1P_n$;
- Ποια σημεία είναι αριστερά ή δεξιά μιας ευθείας;




Μπορούμε σε σταθερό χρόνο *O*(1) να βρούμε την απόσταση ενός σημείου από την ευθεία και να ελέγξουμε και τη μεριά στην οποία βρίσκεται!

- Το πρόσημο της ορίζουσας είναι θετικό μόνο εάν το σημείο $P_3$ είναι στα αριστερά της ευθείας $P_1P_2$;