Εισαγωγη στον Σχεδιασμο και Αναλυση αλγοριθμων
Αλγοριθμοι Ταξινομησης

Η ταξινόμηση μιας ακολουθίας αριθμών είναι από τα βασικά αποτελέσματα της θεωρίας αλγορίθμων. Μια ευρεία γκάμα τέτοιων αλγορίθμων έχουν αναπτυχθεί και εδώ παρουσιάζουμε τους κύριους και ταχύτερους από αυτούς.

Προσοχή: Οι σημειώσεις αυτές αποσκοπούν απλά στην συνολική καταγραφή των ψευδοκωδίκων για να μπορούν οι ενδιαφερόμενοι να έχουν εύκολα πρόσβαση σε αυτούς. Δεν γίνεται επεξήγηση της λειτουργίας τους. Για την κατανόηση της λειτουργίας τους διατρέξτε στην βιβλιογραφία.

Είσοδος: Μια ακολουθία n αριθμών $A =< a_1, a_2, \ldots, a_n >$.
Έξοδος: Μια αναδιάταξη των αριθμών $< a'_1, a'_2, \ldots, a'_n >$ τέτοια ώστε $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

1. **InsertionSort**(**A**) (Ενθετική Ταξινόμηση)
   1. **για** $j = 2$ έως μήκος [A]
   2. κλειδί=A[j]
   3. $i = j - 1$
   4. **ενόσω** $i > 0$ και $A[i] >$ κλειδί
   5. $A[i + 1] = A[i]$
   6. $i = i - 1$
   7. $A[i + 1]$=κλειδί

   Η επίδοση της Ενθετικής ταξινόμησης εξαρτάται άμεσα από τον βαθμό της ταξινόμησης που έχει η είσοδος. Για μια ήδη ταξινομημένη είσοδο θα πάρει χρόνο γραμμικό (που είναι το καλύτερο που μπορούμε να ελπίζουμε μιας και τόσος χρόνος χρειάζεται απλά και μόνο για να διαβάσει τα στοιχεία) αλλά για μια είσοδο που είναι αντίστροφα ταξινομημένη ο αλγόριθμος κάνει τον κακό (αν και όχι τραγικό) χρόνο της τάξης του $\Theta(n^2)$. Για περισσότερες πληροφορίες για τον χρόνο του αλγορίθμου ανατρέξτε (εκτός από το βιβλίο) και στο *google*.

2. **MergeSort**$(\mathbf{A}, \mathbf{p}, \mathbf{r})$ (Συγχωνευτική ταξινόμηση)
   1. **αν** $p < r$
   2. **τότε** $q = \lfloor (p+r)/2 \rfloor$
   3. **MergeSort**$(\mathbf{A}, \mathbf{p}, \mathbf{q})$
   4. **MergeSort**$(\mathbf{A}, \mathbf{q}+\mathbf{1}, \mathbf{r})$
   5. **Merge**(A, p, q, r)

Merge(A, p, q, r) (Συγχώνευση)
1. $n_1 = q - p + 1$
2. $n_2 = r - q$
3. **για** $i = 1$ έως n_1
4. $L[i] = A[p + i - 1]$
5. **για** $j = 1$ έως n_2
6. $R[j] = A[q + j]$
7. $L[n_1 + 1] = \infty$
8. $R[n_2 + 1] = \infty$
9. $i = 1$
10. $j = 1$
11. **για** $k = p$ έως r
12. **αν** $L[i] \leq R[j]$
13. **τότε** $A[k] = L[i]$
14. $i = i + 1$
15. **αλλιώς** $A[k] = R[j]$
16. $j = j + 1$

Ο αλγόριθμος της Συγχωνευτικής Ταξινόμησης είναι κλασσικό παράδειγμα αλγορίθμου που χρησιμοποιεί την τεχνική ''Διαίρει και Βασίλευε''. Ο αλγόριθμος σπάει την ακολουθία στην μέση και δημιουργεί δύο προβλήματα με μέγεθος μισό του αρχικού. Αφού ταξινομίσει αυτά (με την ίδια μέθοδο, δηλαδή με το σπάσει και αυτά στην μέση) μετά καλεί την διαδικασία **Merge**(A, p, q, r) η οποία συγχωνέυει της δύο ταξινομημένες ακολουθίες μισού μεγέθους σε μία!
Η υπορουτίνα **Merge**(A, p, q, r) έχει γραμμικό χρόνο εκτέλεσης, δηλαδή $\Theta(n)$.
Η αναδρομική σχέση που περιγράφει τον χρόνο εκτέλεσης της

MergeSort$(\mathbf{A}, \mathbf{p}, \mathbf{r})$ είναι
$\mathrm{T}(n) = 2T(n/2)+\Theta(n) = O(nlogn)$ (από κεντρικό Θεωρήμα).

3. **QuickSort**$(\mathbf{A}, \mathbf{p}, \mathbf{r})$ (Ταχυταξινόμηση)
   1. **αν** $p < r$
   2. **τοτε** $q =$ **Partition**$(\mathbf{A}, \mathbf{p}, \mathbf{r})$
   3. **QuickSort**$(\mathbf{A}, \mathbf{p}, \mathbf{q} - \mathbf{1})$
   4. **QuickSort**$(\mathbf{A}, \mathbf{q} + \mathbf{1}, \mathbf{r})$

Partition$(\mathbf{A}, \mathbf{p}, \mathbf{r})$ (Διαμέριση)
1. $x = A[r]$
2. $i = p - 1$
3. **για** $j = p$ **έως** $r - 1$
4. **αν** $A[j] \leq x$
5. **τότε** $i = i + 1$
6. **Εναλλαγή**$(A[i], A[j])$
7. **Εναλλαγή**$(A[i + 1], A[r])$
8. **επιστροφή**$(i + 1)$

Η υπορουτίνα $Partition(A, p, r)$ έχει χρόνο εκτέλεσης γραμμικό, δηλαδή $\Theta(n)$.
Η επίδοση της ταχυταξινόμησης εξαρτάται άμεσα από την διαμερίση που θα παράξει η υπορουτίνα $Partition(A, p, r)$.

Στην χειρότερη περίπτωση θα μας δώσει ένα υποπρόβλημα μήκους $n - 1$ και ένα μήκους 0 οπότε η αναδρομική σχέση θα είναι:
$\mathrm{T}(n)=\mathrm{T}(n-1)+\mathrm{T}(0)+\Theta(n)=\Theta(n^2)$.
Όχι και τόσο καλός χρόνος.

Στην καλή περίπτωση όμως η διαμέριση γίνεται στη μέση της ακολουθίας και τότε η αναδρομική σχέση γίνεται
$\mathrm{T}(n) \leq 2T(n/2)+\Theta(n)=O(nlogn)$ (από το κεντρικό θεώρημα).

Το ενδιαφέρον στον αλγόριθμο αυτό είναι πως ο χρόνος παραμένει ίδιος ακόμα και αν η διαμέριση δεν γίνεται στην μέση αλλά απλά ισομερώς, αρκεί αυτό να συμβαίνει σε κάθε επανάληψη της υπορουτίνας $Partition(A, p, r)$. Δηλαδή ο αλγόριθμος τρέχει κοντά στον χρόνο καλύτερης περίπτωσης ακόμα και όταν η είσοδος είναι τέτοια που διαμερίζεται συνέχεια σε υποπροβλήματα με αναλογία 9 προς 1 (που διασθητικά φαίνεται πολύ κακός). Έτσι η επίδοση μέσης περίπτωσης είναι πολύ κοντά στην επίδοση καλύτερης περίπτωσης (αλλιώς δεν θα τον αναφέραμε καν).

4. **HeapSort**(**A**) (Ταξινόμηση Σωρού)
   1. **CreateMaxHeap**(**A**)
   2. **για** $i =$ μήκος [A] **αντίστροφα έως** 2
   3. **Εναλλαγή**(A[1],A[i])
   4. ΠλήθοσΣωρού(A)=ΠλήθοσΣωρού(A)-1
   5. **RestoreMaxHeap**(**A**, **1**)

RestoreMaxHeap(**A**, **i**) (Αποκατάσταση Σωρού Μεγίστου)
1. $l =$Αριστερός (i)
2. $r =$Δεξιός (i)
3. **αν** $l \leq$ ΠλήθοσΣωρού(A) και $A[l] > A[i]$
4. **τότε** #-μεγίστου=l
5. **αλλιώς** #-μεγίστου=i
6. **αν** $r \leq$ ΠλήθοσΣωρού(A) και $A[r] > A[$#-μεγίστου$]$
7. **τότε** #-μεγίστου=r
8. **αν** #-μεγίστου $\neq i$
9. **τότε** Εναλλαγή$(A[i], A[$#-μεγίστου$])$
10. **RestoreMaxHeap**(**A**, #$-$μεγίστου)

CreateMaxHeap(**A**) (Κατασκευή Σωρού Μεγίστου)
1. ΠλήθοσΣωρού[A]=μήκος (A)
2. **για** $i = \lfloor$μήκος (A)$/2 \rfloor$ αντίστροφα έως 1
3. **RestoreMaxHeap**(**A**, **i**)

Οι υπορουτίνες **Αριστερός** i) και **Δεξιός** i) επιστρέφουν αντίστοιχα $2i$ και $2i + 1$.
Ο αλγόριθμος της ταξινόμησης σωρού χρησιμοποιεί (τι άλλο!) μια σωρό. Για τον ορισμό και τις ιδιότητες μια σωρού ανατρέξτε στην βιβλιογραφία και στο διαδίκτιο.
Η διαδικασία **RestoreMaxHeap**(**A**, **1**) απαιτεί λογραιθμικό χρόνο και είναι εύκολο να δει κανείς πως καλείτε n φορές. Άρα συνολικά $nlogn$. Τόσο χρόνο απαιτεί και η εκτέλεση της διαδικασίας **CreateMaxHeap**(**A**) η οποία εκτελείται μόνο μια φορά. Τελικά λοιπόν έχουμε πως
$\mathrm{T}(n) = O(nlogn) + nlogn = O(nlogn)$

5. **Γενικές Παρατηρήσεις:**

(α΄) Έχει αποδειχθεί πως κάθε αλγόριθμος ταξινόμησης που χρησιμοποιεί συγκρίσεις πρέπει να πάρει χρόνο $\Omega(nlogn)$ (Είναι λίγο πιο προχωρημένο, αλλά είναι μέσα στις δυνατότητές σας να το αποδείξετε αυτό). Δηλαδή οι αλγόριθμοι **MergeSort**(**A**) και **HeapSort**(**A**) είναι βέλτιστοι! Αποκλείεται να βρεθεί αλγόριθμος πιο γρήγορος από αυτούς.

(β΄) Παρ΄ όλα αυτά υπάρχουν κάποιοι αλγόριθμοι που λειτουργούν υπό ορισμένες προυποθέσεις και τρέχουν σε χρόνο γραμμικό.

(γ΄) Όπως επανειλημμένα ειπώθηκε παραπάνω, για περισσότερες πληροφορίες και κατανόηση των αλγορίθμων αυτών ανατρέξτε σε παραδείγματα σε βιβλία και στο διαδίκτυο (και στις παραδόσεις του μαθήματος φυσικά!).