Διάλεξη 18 - Ταξινόμηση #2

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

Ανακοινώσεις / Διευκρινήσεις

- Ευχαριστούμε τον Πέτρο για τις οπτικοποιήσεις των αλγορίθμων ταξινόμησης!
- Βγήκε η Εργασία #2 (επιτέλους!)
- Η Εργασία #1 είναι πίσω μας (ευτυχώς)

Την προηγούμενη φορά

- Αλγόριθμοι Αναζήτησης (Search Algorithms)
  - Γραμμική Αναζήτηση (Linear Search)
  - Δυαδική Αναζήτηση (Binary Search)
- Αλγόριθμοι Ταξινόμησης (Sorting Algorithms)

Σήμερα

- Αλγόριθμοι Ταξινόμησης (Sorting Algorithms)
- Παραδείγματα

Δυαδική Αναζήτηση (Binary Search)

```c
int binary_search(int elem, int *array, int n) {
  int mid, low = 0, high = n - 1;
  while (low <= high) {
    mid = low + (high - low) / 2;
    if (array[mid] == elem)
      return 1;
    else if (array[mid] < elem)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return 0;
}
```

Τι πολυπλοκότητα έχει αυτός ο αλγόριθμος;

Χρόνος: O(logn)
Χώρος: O(1)

Υλοποιημένη στην συνάρτηση `bsearch` της stdlib.h

Αλγόριθμοι Ταξινόμησης (Sorting Algorithms)

1. Bubblesort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quicksort

Γράψτε μια συνάρτηση swap που ανταλλάζει δύο ακεραίους

```c
#include <stdio.h>
int main() {
  int a = 100, b = 200;
  printf("%d %d\n", a, b);
  swap( ... );
  printf("%d %d\n", a, b);
  return 0;
}
```

Γράψτε μια συνάρτηση swap που ανταλλάζει δύο ακεραίους

```c
#include <stdio.h>

void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main() {
  int a = 100, b = 200;
  printf("%d %d\n", a, b);
  swap(&a, &b);
  printf("%d %d\n", a, b);
  return 0;
}
```

Ταξινόμηση Επιλογής (Selection Sort)

```
void selection_sort(int n, int *x) {
  int i, j, min;
  for (i = 1 ; i <= n - 1 ; i++) {
    min = i - 1;
    for (j = i ; j <= n - 1 ; j++)
      if (x[j] < x[min])
        min = j;
    swap(&x[i-1], &x[min]);
  }
}
```

Ταξινόμηση Επιλογής (Selection Sort)

```
void selection_sort(int n, int *x) {
  int i, j, min;
  for (i = 1 ; i <= n - 1 ; i++) {
    min = i - 1;
    for (j = i ; j <= n - 1 ; j++)
      if (x[j] < x[min])
        min = j;
    swap(&x[i-1], &x[min]);
  }
}
```

Χρόνος: $O(n^2)$
Χώρος: $O(1)$

Ταξινόμηση Εισαγωγής (Insertion Sort)

```
void insertion_sort(int n, int *x) {
  int i, j;
  for (i = 1 ; i <= n - 1 ; i++) {
    j = i - 1;
    while (j >= 0 && x[j] > x[j+1]) {
      swap(&x[j], &x[j+1]);
      j--;
    }
  }
}
```

Ταξινόμηση Εισαγωγής (Insertion Sort)

```
void insertion_sort(int n, int *x) {
  int i, j;
  for (i = 1 ; i <= n - 1 ; i++) {
    j = i - 1;
    while (j >= 0 && x[j] > x[j+1]) {
      swap(&x[j], &x[j+1]);
      j--;
    }
  }
}
```

Χρόνος: $O(n^2)$
Χώρος: $O(1)$

Ταξινόμηση Φυσαλίδας (Bubblesort)

```
void bubblesort(int n, int *x) {
  int i, j;
  for (i = 1 ; i <= n - 1 ; i++)
    for (j = n - 1 ; j >= i ; j--)
      if (x[j-1] > x[j])
        swap(&x[j-1], &x[j]);
}
```

Ταξινόμηση Φυσαλίδας (Bubblesort)

```
void bubblesort(int n, int *x) {
  int i, j;
  for (i = 1 ; i <= n - 1 ; i++)
    for (j = n - 1 ; j >= i ; j--)
      if (x[j-1] > x[j])
        swap(&x[j-1], &x[j]);
}
```

Χρόνος: $O(n^2)$
Χώρος: $O(1)$

Ταξινόμηση Συγχώνευσης (Merge Sort)

Η ταξινόμηση συγχώνευσης (merge sort) είναι ένας αλγόριθμος divide and conquer (διαίρει και βασίλευε) που έχει θεωρητικά την καλύτερη πολυπλοκότητα. Ο αλγόριθμος έχει δύο βήματα:

1. Χώρισε τον πίνακα σε δύο υποπίνακες
    a. κάλεσε ταξινόμηση συγχώνευσης στους υποπίνακες
2. Συγχώνευσε τα στοιχεία των δύο ταξινομημένων υποπινάκων

Ταξινόμηση Συγχώνευσης (Merge Sort)

```
void merge_sort(int *array, int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;
        merge_sort(array, left, middle);
        merge_sort(array, middle + 1, right);
        merge(array, left, middle, right);
    }
}
```

Ταξινόμηση Συγχώνευσης (Merge Sort)

```
void merge_sort(int *array, int left, int right) {
    if (left < right) {
        int middle = left + (right - left) / 2;
        merge_sort(array, left, middle);
        merge_sort(array, middle + 1, right);
        merge(array, left, middle, right);
    }
}
```

Ταξινόμηση Συγχώνευσης (Merge Sort)

```
void merge(int *x, int l, int m, int r) {
    int i, j, k, n1 = m - l + 1, n2 = r - m;
    int left[n1], right[n2];
    for (i = 0; i < n1; i++) left[i] = x[l + i];
    for (j = 0; j < n2; j++) right[j] = x[m + 1 + j];
    i = 0; j = 0; k = l;
    while (i < n1 && j < n2) {
        if (left[i] <= right[j]) x[k++] = left[i++];
        else x[k++] = right[j++];
    }
    while (i < n1) x[k++] = left[i++];
    while (j < n2) x[k++] = right[j++];
}
```

Ταξινόμηση Συγχώνευσης (Merge Sort)

```
void merge(int *x, int l, int m, int r) {
    int i, j, k, n1 = m - l + 1, n2 = r - m;
    int left[n1], right[n2];
    for (i = 0; i < n1; i++) left[i] = x[l + i];
    for (j = 0; j < n2; j++) right[j] = x[m + 1 + j];
    i = 0; j = 0; k = l;
    while (i < n1 && j < n2) {
        if (left[i] <= right[j]) x[k++] = left[i++];
        else x[k++] = right[j++];
    }
    while (i < n1) x[k++] = left[i++];
    while (j < n2) x[k++] = right[j++];
}
```

Χρόνος: O(nlogn)
Χώρος: O(n)

Ταχυταξινόμηση (Quicksort)

Η ταξινόμηση ταχυταξινόμηση (quicksort) είναι ένας αλγόριθμος divide and conquer (διαίρει και βασίλευε) που είναι ιδιαίτερα δημοφιλής. Ο αλγόριθμος έχει τρία βήματα:

1. Διάλεξε (έστω τυχαία) το στοιχείο διαμέρισης του πίνακα (pivot element)
2. Διαμέρισε τον πίνακα σε δύο υποπίνακες - αριστερά έχει τα στοιχεία που είναι μικρότερα του pivot και δεξιά τα στοιχεία που είναι μεγαλύτερα
3. Τρέξε ταχυταξινόμηση για τους δύο υποπίνακες

Ταχυταξινόμηση (Quicksort)

```
void quicksort (int *x, int lower, int upper) {
   if (lower < upper) {
      int pivot = x[(lower + upper) / 2];
      int i, j;
      for (i = lower, j = upper; i <= j;) {
        while (x[i] < pivot) i++;
        while (x[j] > pivot) j--;
        if (i <= j) swap(&x[i++], &x[j--]);
        }
      quicksort(x, lower, j);
      quicksort(x, i, upper);
   }
 }
```

Ταχυταξινόμηση (Quicksort)

```
void quicksort (int *x, int lower, int upper) {
   if (lower < upper) {
      int pivot = x[(lower + upper) / 2];
      int i, j;
      for (i = lower, j = upper; i <= j;) {
        while (x[i] < pivot) i++;
        while (x[j] > pivot) j--;
        if (i <= j) swap(&x[i++], &x[j--]);
        }
      quicksort(x, lower, j);
      quicksort(x, i, upper);
   }
 }
```

Χρόνος: $O(n^2)$ (worst case), O(nlogn) (average case)
Χώρος: O(n) (εδώ) - γίνεται και σε O(logn)

Υλοποιημένη στην συνάρτηση `qsort` της stdlib.h

Ένα Παράδειγμα

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char *months[] = {"jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep", "oct", "nov",
"dec"};

#define nr_of_months (sizeof(months) / sizeof(months[0]))

int comp_months(const void *m1, const void *m2) {

  return strcmp(*(const char **)m1, *(const char **)m2);

}

int main(int argc, char **argv) {

  qsort(months, nr_of_months, sizeof(char *), comp_months);

  char **res = (char**)bsearch(&argv[1], months, nr_of_months, sizeof(char *), comp_months);

  if (res == NULL) printf("'%s': unknown month\n", argv[1]);

  else printf("%s: is a month\n", *res);

  return 0;

}
```

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα ακεραίων και έναν ακέραιο g και να επιστρέφει αν υπάρχουν δύο ακέραιοι του πίνακα που αθροίζουν στον g (two sum). Πως;

Θέλω να γράψω μια συνάρτηση που υπολογίζει την ρίζα ενός ακεραίου n χωρίς χρήση της math.h (sqrt). Πως;

Θέλω να δημιουργήσω έναν δισδιάστατο πίνακα ακεραίων MxN δυναμικά. Πως;

```
int ** array = malloc(M * sizeof(int*));
if (!array) {
    perror("array allocation");
    exit(1);
}
for(int i = 0 ; i < M ; i++) {
    array[i] = malloc(N * sizeof(int));
    if (!array[i]) {
        perror("array[i] failed");
        exit(1);
    }
}
```

Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου καλύψαμε τις σελίδες 160-177,

- Διαφορετικοί τρόποι να δεσμεύσεις μνήμη για δισδιάστατο πίνακα

Ευχαριστώ και καλό σαβ/κο εύχομαι!
Keep Coding ;)