חלק 2.1 – רשימות מקושרות

1. גרף – מבנה הנתונים של גרף מכיל בתוכו מידע על האוביקטים (יעדי הטיסה) ועל הקשרים ביניהם (מסלולי הטיסה). לכן בעזרת מבנה זה יהיה נוח לשמור את נתוני הבעיה.
2. מחסנית – בעזרת מחסנית ניתן לשמור את סדר השינויים בתוכנה בצורה קלה, וביטול הפעולה האחרונה (פעולת undo) דורש פשוט הוצאה של האיבר האחרון שהוכנס למחסנית
3. רשימה – באמצעות רשימה ניתן לשמור כמות בלתי מוגבלת של הזמנות, תוך כדי שמירה על הסדר ביניהן, אשר צריך להתנהל בצורה הוגנת (הראשון שהזמין אוכל יקבל את מנתו ראשון)
4. קבוצה – מבנה של קבוצה לא יכול להכיל כפילויות, ולכן שימוש במבנה זה חוסך מימוש של פונקציונליות זו, ומוודא שלא יהיה אותו תחביב פעמיים לאותו אדם.

חלק 2.2 – מיון גנרי

*/\*\*  
 \* swap - switches between 2 elements in the array  
 \* @param a - the first element to switch  
 \* @param b - the second element to switch  
 \*/***void** swap(**void**\*\* a, **void**\*\* b) {  
 **void**\* temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
}  
  
*/\*\*  
 \* partition - places the pivot element (in high index) at it's correct place and split the elements in the array  
 \* between the elements that are bigger than pivot element and elements that are smaller than pivot element  
 \* @param array - the array to sort  
 \* @param low - the index of the first element in the region the function splits  
 \* @param high - the index number of the pivot element  
 \* @param compareFunction - the function which compare between two elements  
 \* @return  
 \* the index of the correct place of the pivot number  
 \*/***int** partition(**void**\*\* array, **int** low, **int** high, **int** (\*compareFunction)(**void**\*, **void**\*)) {  
 **void**\* pivot = array[high];  
 **int** i = low - 1;  
  
 **for** (**int** j = low; j <= high -1; j++) {  
 **if** (compareFunction(array[j], pivot) <= 0) {  
 i++;  
 swap(&array[i], &array[j]);  
 }  
 }  
 swap(&array[i+1], &array[high]);  
 **return** i+1;  
}  
  
*/\*\*  
 \* quickSortAux - sorts the array in the given region (between low and high) from smaller to bigger (according to the  
 \* compare function given)  
 \* @param array - the array to sort  
 \* @param low - the lower boundary in the sorted region  
 \* @param high - the upper boundary in the sorted region  
 \* @param compareFunction - the function which compare between two elements  
 \*/***void** quickSortAux(**void**\*\* array, **int** low, **int** high, **int** (\*compareFunction)(**void**\*, **void**\*)) {  
 **if** (low < high) {  
 **int** partition\_index = partition(array, low, high, compareFunction);  
 quickSortAux(array, low, partition\_index - 1, compareFunction);  
 quickSortAux(array, partition\_index + 1, high, compareFunction);  
 }  
}  
  
*/\*\*  
 \* quickSort - sorts the array from smaller to bigger (according to the compare function given)  
 \* @param array - the array to sort  
 \* @param size - the size of the array  
 \* @param compareFunction - the function which compare between two elements  
 \*/***void** quickSort(**void**\*\* array, **int** size, **int** (\*compareFunction)(**void**\*, **void**\*)) {  
 **int** low = 0;  
 **int** high = size - 1;  
 quickSortAux(array, low, high, compareFunction);  
}

חלק 2.3 – רשימות מקושרות

1. הפעולה listCopy:



2. הפעולה listCopyReversed:



3. הפעולה listJoinAlternating:

