1. Зробити оцінку складності алгоритму за кодом

```
void func(int[] A, int low, int high)
{
	int i = low;
	int j = high;
	int x = A[(low+high)/2];
	do {
		while(A[i] < x) ++i;
		while(A[j] > x) --j;
		if(i <= j)
		{
			int temp = A[i];
			A[i] = A[j];
			A[j] = temp;
			i++; j--;
		}
	}
	while(i < j);

	if(low < j) //recursion
		func(A, low, j);

	if(i < high) //recursion
		func(A, i, high);
}
```

Функція реалізує quicksort з розбиттям Хоара.

Найкращий випадок – якщо кожен субмасив ділиться на два збалансованих субмасиви.
Складність: $O(nlog_2(n))$

Середній випадок – елементи розташовані рандомно.
Складність: $O(nlog(n))$

Найгірший випадок – якщо при кожному діленні на субмасиви опорним елементом буде вибране мінімальне або максимальне значення.
Складність: $O(n^2)$

2. Виберіть найбільш підходящий контейнер для збереження інформації телефонного довідника. Мається на увазі, що довідник дозволяє знайти інформацію про абонента за номером. Обґрунтуйте вибір певного контейнера.

Обґрунтування передбачає приведення списку операцій, що найбільш часто застосовуються для цього контейнера, і оцінок їх складностей по О нотації.

Операції в довіднику (за частотою використання):

1. Пошук номера за ім'ям абонента (Find);
2. Додавання контакта (Insert);
3. Коригування контакта (Find);
4. Видалення контакта (Find+Remove).

Емпіричний аналіз вказує на те, що операція Find одна з найчастіше використовуваних. Також візьмемо до уваги той факт, що телефонний довідник зберігає дані у форматі ключ+значення (контакт абонента + номер).

Отже, згідно таб.1, слід використати **map** або **multimap** (якщо дозволяється дублювання контактної інформації). Пошук, вставка, видалення виконується за $O(log(n))$.

Container	Insert Head	Insert Tail	**Insert**	Remove Head	Remove Tail	**Remove**	Index Search	Find
vector	n/a	O(1)	O(n)	O(1)	O(1)	O(n)	O(1)	O(log n)
list	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	n/a	O(n)
deque	O(1)	O(1)	n/a	O(1)	O(1)	O(n)	n/a	n/a
queue	n/a	O(1)	n/a	O(1)	n/a	n/a	O(1)	O(log n)
stack	O(1)	n/a	n/a	O(1)	n/a	n/a	n/a	n/a
map	n/a	n/a	O(log n)	n/a	n/a	O(log n)	O(1)	O(log n)
multimap	n/a	n/a	O(log n)	n/a	n/a	O(log n)	O(1)*	O(log n)
set	n/a	n/a	O(log n)	n/a	n/a	O(log n)	O(1)	O(log n)
multiset	n/a	n/a	O(log n)	n/a	n/a	O(log n)	O(1)*	O(log n)

Таблиця 1. Складність операцій в STL контейнерах. (link)

Ще однією перевагою вибору map буде ефективність функції автокомпліту пошуку за ім'ям абонента.

Також відмітимо гнучкість імплементації композитного ключа з відповідним компаратором. Можна сформувати ключі з полів самого контакта (прізвище+ім'я; номер), (прізвище; номери*), (місто+вулиця+прізвище; номери*). Звісно доведеться мати копію відповідно відбалансованого дерева. Тобто ми жертвуємо стораджем заради гнучкості і швидкості, але у відповідних випадках це може бути виправданим.

*при використанні мультимапи або можна повертати всі кінечні ліфи з субдерева.