```
// Ordena um vetor de indice inicial p e final r
void insertionsort(int *A, int p, int r)
{
	for(int i=p+1; i<=r; i++)
	{
		int j = i;

		while(j>p && A[j-1] > A[j])
		{
			permutar(A, j, j-1);
			j=j-1;
		}
	}
}

// Ordena um vetor de indice inicial p e final r
void quicksort(int *A, int p, int r, int m)
{
	if(r-p<=m)
	{
		insertionsort(A, p, r);
	}
	else
	{
		int q = particionarUsandoPivoAleatorio(A, p, r);
		quicksort(A, p, q-1, m);
		quicksort(A, q+1, r, m);
	}
}

// escolhe um pivô aleatório para evitar o pior caso do quicksort
int particionarUsandoPivoAleatorio(int *A, int p, int r)
{
	// Atualiza a semente
	semente = semente+43;
	srand(semente);

	// seleciona um número entre r (inclusive) e p (inclusive)
	int pivo_indice = (rand() % (r - p + 1)) + p;

	// faz a troca para colocar o pivô no r
	permutar(A, pivo_indice, r);

	// chama a particionar
	return particionar(A, p, r);
}
```

```
// Particiona um vetor considerando o ultimo elemento pivô
int particionar(int *A, int p, int r)
{
	int pivo = A[r];
	int i = p-1;

	for(int j=p; j<=r-1; j++)
	{
		// Incrementa comparacoes
		comparacoes++;

		if(A[j] <= pivo)
		{
			i=i+1;
			permutar(A, i, j);
		}
	}
	permutar(A, i+1, r);
	return i+1;
}

// Permuta os elementos de indice i e j num vetor dado
void permutar(int *vetor, int i, int j)
{
	// Permuta dois elementos
	int x = vetor[i];
	vetor[i] = vetor[j];
	vetor[j] = x;

	// Incrementa trocas
	trocas++;
}
```