

Binary Search

int binarySearch(int array[], int size, int value)

{

int first = 0, // First array element

last = size - 1, // Last array element

middle, // Mid point of search

position = -1; // Position of search value

bool found = false; // Flag

while (!found && first <= last)

{

middle = (first + last) / 2; // Calculate mid point

if (array[middle] == value) // If value is found at mid

{

found = true;

position = middle;

}

else if (array[middle] > value) // If value is in lower half

last = middle - 1;

else

first = middle + 1; // If value is in upper half

}

return position;

}

Linear Search Algorithm

int searchList(int list[], int numElems, int value)

{

int index = 0; // Used as a subscript to search array

int position = -1; // To record position of search value

bool found = false; // Flag to indicate if value was found

while (index < numElems && !found)

{

if (list[index] == value) // If the value is found

{

found = true; // Set the flag

position = index; // Record the value's subscript

}

index++; // Go to the next element

}

return position; // Return the position, or -1

}

Below is the basic bubble sort algorithm.

**void** bubbleSort(**int** numbers[], **int** array\_size)

{

**int** i, j, temp;

**for** (i = (array\_size - 1); i >= 0; i--)

{

**for** (j = 1; j <= i; j++)

{

**if** (numbers[j-1] > numbers[j])

{

temp = numbers[j-1];

numbers[j-1] = numbers[j];

numbers[j] = temp;

}

}

}

}

**void** insertionSort(**int** numbers[], **int** array\_size)

{

**int** i, j, index;

**for** (i=1; i < array\_size; i++)

{

index = numbers[i];

j = i;

**while** ((j > 0) && (numbers[j-1] > index))

{

numbers[j] = numbers[j-1];

j = j - 1;

}

numbers[j] = index;

}

}

Merge sort

**void** mergeSort(**int** numbers[], **int** temp[], **int** array\_size)

{

m\_sort(numbers, temp, 0, array\_size - 1);

}

**void** m\_sort(**int** numbers[], **int** temp[], **int** left, **int** right)

{

**int** mid;

**if** (right > left)

{

mid = (right + left) / 2;

m\_sort(numbers, temp, left, mid);

m\_sort(numbers, temp, mid+1, right);

merge(numbers, temp, left, mid+1, right);

}

}

**void** merge(**int** numbers[], **int** temp[], **int** left, **int** mid, **int** right)

{

**int** i, left\_end, num\_elements, tmp\_pos;

left\_end = mid - 1;

tmp\_pos = left;

num\_elements = right - left + 1;

**while** ((left <= left\_end) && (mid <= right))

{

**if** (numbers[left] <= numbers[mid])

{

temp[tmp\_pos] = numbers[left];

tmp\_pos = tmp\_pos + 1;

left = left +1;

}

**else**

{

temp[tmp\_pos] = numbers[mid];

tmp\_pos = tmp\_pos + 1;

mid = mid + 1;

}

}

**while** (left <= left\_end)

{

temp[tmp\_pos] = numbers[left];

left = left + 1;

tmp\_pos = tmp\_pos + 1;

}

**while** (mid <= right)

{

temp[tmp\_pos] = numbers[mid];

mid = mid + 1;

tmp\_pos = tmp\_pos + 1;

}

**for** (i=0; i <= num\_elements; i++)

{

numbers[right] = temp[right];

right = right - 1;

}

}

Below is the basic quick sort algorithm.

**void** quickSort(**int** numbers[], **int** array\_size)

{

q\_sort(numbers, 0, array\_size - 1);

}

**void** q\_sort(**int** numbers[], **int** left, **int** right)

{

**int** pivot, l\_hold, r\_hold;

Página 14 de 18

l\_hold = left;

r\_hold = right;

pivot = numbers[left];

**while** (left < right)

{

**while** ((numbers[right] >= pivot) && (left < right))

right--;

**if** (left != right)

{

numbers[left] = numbers[right];

left++;

}

**while** ((numbers[left] <= pivot) && (left < right))

left++;

**if** (left != right)

{

numbers[right] = numbers[left];

right--;

}

}

numbers[left] = pivot;

pivot = left;

left = l\_hold;

right = r\_hold;

**if** (left < pivot)

q\_sort(numbers, left, pivot-1);

**if** (right > pivot)

q\_sort(numbers, pivot+1, right);

}

Below is the basic selection sort algorithm.

**void** selectionSort(**int** numbers[], **int** array\_size)

{

**int** i, j;

**int** min, temp;

**for** (i = 0; i < array\_size-1; i++)

{

min = i;

**for** (j = i+1; j < array\_size; j++)

{

**if** (numbers[j] < numbers[min])

min = j;

}

temp = numbers[i];

numbers[i] = numbers[min];

numbers[min] = temp;