

# PUSH - SWAP ( )

## Parrafo

// leer\_args.c

```
void liberar_array(char *array);  
int free_stack(t_stack *s);  
int list_nums(char const *str);  
t_stack init_stack(int *num, int size);  
t_list *new_stack_node(int n);
```

## Comprobaciones

// check.c

```
int checkOrder(t_list *ord);  
int noRep(char *argv, int argc);  
int checkNum(char *argv, int argc);
```

## UTILID

// utilD.c

```
int get_index(int *arr, int size, int val);  
int get_pos(t_list *list, int val);  
t_list *listnum(t_list *list, int num);  
int minlist(t_list *list);  
void ft_printlist(t_list *a);
```

## Ordenación

// sortone

```
void sortOne(t_list *a);  
void sortTwo(t_list *a);  
void sort-four-or-five(t_stack *s);  
void selector(t_stack *stack);
```

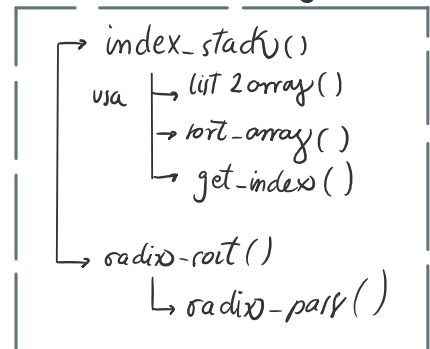
// algorithm.c

```
void radix-sort(t_list **a, t_list **b);  
void radix-part(t_list **a, t_list **b,  
                int i, int size);  
void index-stack(t_list *a);  
int *list2array(t_list *a, int size);  
void sort_array(int *arr, int size);
```

## Estructura de datos

```
typedef struct s_stack  
{  
    t_list *a;  
    t_list *b;  
}  
// linked list  
typedef struct s_list  
{  
    void *content;  
    struct s_list *next;  
}  
t_list
```

## Algoritmo



# Flujo del main

```
int main(int argc, char *argv[])
```

## Control del input

Comprobar que se hayan pasado argumentos  
 if (argc < 2)  
 return 0;

usa  $\times$  ft\_strlen()

```
if (!checknum(argc, argv) || !norep(argv, argc))
```

```
int norep(char *argv, int argc);
```

comprueba que no hayan dígitos repetidos

```
int checknum(char *argv, int argc);
```

Controla el input (que sean números enteros)

usa  $\downarrow$   
 ft\_strdup()

## Parseo de argumentos

```
[./push_swap]  
[53]  
[6]  
[7]
```

```
int list_nums(char *argv, int size);
```

convierte el array a una lista

ft\_atoi

[53, 6, 7]

```
t_stack_t init_stack(t *num, int size);
```

- Inicializa ambos listas en el stack NULL
- para cada número del array, crea un nodo de la linked list  $\times$  new\_stack\_node (num[i])
- Si todo va bien anexa el nodo al final de la lista  $\times$  ft\_ladd-back (&ra, node)

cada posición del array

s.a | s.b  
 53 | NULL  
 6 |  
 7 |  
 NULL

## Algoritmo de ordenación

```
void selector(t_stack_t *stack);
```

Se llama antes de aplicar radix

```
void index_stack(t_list *a);
```

Radix binario no acepta valores negativos, por eso esta función convierte los valores a índices

5 a -10 -20 0 -30  
 list array [-10, 20, 0, -30]  
 sort array [-30, -10, 0, 20]  
 index stack [1 -> 3 -> 2 -> 0]

ejemplo

bubble sort

hardcoded

```
→ rotch(t_list *a);  
→ rotch(t_list *a);  
→ sort_four_or_five(t_stack_t *s);  
→ radix_rot(t_list **a, t_list **b);  
→ radix_parr(t_list **a, t_list **b, int i, int size);
```

max\_bits:  
 cuantos bits son necesarios para representar al número más grande de la lista

```
radix_rot(t_list **a, t_list **b)
```

ordena a usando radix

- Verifica si ya está ordenado
- Calcula el tamaño de la lista
- Calcula max\_bits while ((size-1) >> max\_bits)
- desplaza (size-1) a la derecha bit a bit
- por cada desplazamiento incrementa max\_bits
- Empieza radix\_parr(a, b, i, size)
- mueve los bits que tienen el índice i = 0 hacia b
- los que tienen 1 los deja en a
- al terminar la iteración, mover todo de b a a
- comprobar si ya está ordenado

>>> operador right shift  
 es como descomponer en factores primos

8 >> 1 = 4  
 8 >> 2 = 2  
 8 >> 3 = 1

cada desplazamiento es quitar un bit a la derecha

con size = 8

7 >> 0 [111] = 7  
 7 >> 1 [110] = 3  
 7 >> 2 [100] = 1  
 7 >> 3 [000] = 0

Ej. 1

Ej. 2