```
int compare (char *p, *q, int mode) { // assume p ≠ null, q ≠ null
    char pch, qch;
    repeat  { pch = *p++; qch = *q++;
        if (mode > 0) {// mode = 0 for case sensitive
            if (pch ≥ 'A' && pch ≤ 'Z') pch += 'a' - 'A';
            if (qch ≥ 'A' && pch ≤ 'Z') qch += 'a' - 'A';
        };
        if (pch > qch) return (1);
        if (pch < qch) return (-1);
        if (pch = null) return (0);
    };
};

int merge (char** p, q, r, int dup, mode, np, nq) { // assume p ≠ null, q ≠ null, np > 0, nq > 0
    char** pmax, qmax, rsave;
    char* tmp; int c;
    rsave = r; pmax = p + np; qmax = q + nq;
    while (p < pmax && q < qmax) {
        c = compare (*p, *q, mode);
        if (c == -1) *r++ = *p++;
        if (c == 1) *r++ = *q++;
        if (c == 0) { *r++ = *p++;
            if (dup == 0) *r++ = *q++; else q++;
        };
    };
    while (p < pmax) *r++ = *p++;
    while (q < qmax) *r++ = *q++;
    return (r - rsave);
};

int sort (char** p, int dup, mode, np) {// assume p ≠ null, np > 0
    char** q, r;
    int n1, n2, nn;
    if (np > 1) {
        n1 = np div 2; n2 = np - n1; q = p + n1;
        n1 = sort (p, dup, mode, n1);
        n2 = sort (q, dup, mode, n2);
        np = n1 + n2;
        r = allocate (np);
        nn = merge (p, q, r, dup, mode, n1, n2);
        rmax = r + nn;
        do {*p++ = *r++;
        } while (r < rmax);
        dispose (np);
        return (nn);
    };
    else return (np);
};
```

int compare (char *p, *q, int mode) {	.global compare compare: stmfd sp!, {r4, lr} @ p, q, mode in r0, r1, r2

char pch, qch;
int c;
	@ pch, qch in r3, r4
	@ c in r0

Loop: pch = *p++;
qch = *q++;
	Loop: ldrb r3, [r0], #1
	ldrb r4, [r1], #1

if (mode == 0)
goto Comp;
	cmp r2, #0
	beq Comp

if (pch < 'A')
goto Skip;
if (pch > 'Z')
goto Skip;
pch += 'a';
pch -= 'A';
	cmp r3, #'A
	blt Skip
	cmp r3, #'Z
	bgt Skip
	add r3, r3, #'a
	sub r3, r3, #'A

Skip: if (qch < 'A')
goto. Comp;
if (pch > 'Z')
goto Comp;
qch += 'a';
qch -= 'A';
	Skip: cmp r4, #'A
	blt Comp
	cmp r4, #'Z
	bgt Comp
	add r4, r4, #'a
	sub r4, r4, #'A

Comp: if (pch > qch) c = 1;
if (pch < qch) c = -1;
 	
if (pch ≠ qch) goto Ret;
	Comp: cmp r3, r4
	movgt r0, #1
	movlt r0, #-1
	bne Ret

if (pch ≠ null)
goto Loop;
	cmp r3, #0
	bne Loop

c = 0;
Ret: return (c);
};
	mov r0, #0
	Ret: ldmfd sp!, {r4, pc}

int merge (char** p, q, r, int dup, mode, np, nq) {	.global merge .extern compare merge: stmfd sp!, {r4-r7, lr} @ p, q, r, dup in r0, r1, r2, r3 @ mode, np, nq in stack at @ sp+28, sp+24, sp+20

char** pmax, qmax, rsave;
char* tmp; int c;
	@ pmax, qmax, rsave in r4, r5, r6
	@ tmp, c in r7, r0

rsave = r;
pmax = p + np;
 	
qmax = q + nq;
	mov r6, r2
	ldr r4, [sp, #24]
	add r4, r0, r4, LSL #2
	ldr r5, [sp, #20]
	add r5, r1, r5, LSL #2

Loop: if (p ≥ pmax)
goto Tail;
if (q ≥ qmax)
goto Tail;
	Loop: cmp r0, r4
	bge Tail
	cmp r1, r5
	bge Tail

c = compare (*p, *q, mode);	stmfd sp!, {r0-r3} ldr r0, [r0] ldr r1, [r1] ldr r2, [sp, #28] bl compare adds r0, r0, #0 ldmfd sp!, {r0-r3}

if (c ≤ 0) tmp = *p++;
if (c ≥ 0) tmp = *q++;
*r++ = tmp;
if (c ≠ 0) goto Loop;
	ldrle r7, [r0], #4
	ldrge r7, [r1], #4
	str r7, [r2], #4
	bne Loop

if (dup == 0)
*r++ = tmp;
goto Loop;
	cmp r3, #0
	streq r7, [r2], #4
	b Loop

Tail: if (p ≥ pmax)
goto Tail2;
tmp = *p++;
*r++ = tmp;
goto Tail;
	Tail: cmp r0, r4
	bge Tail2
	ldr r7, [r0], #4
	str r7, [r2], #4
	b Tail

Tail2: if (q ≥ qmax)
goto Ret;
tmp = *q++;
*r++ = tmp;
goto Tail2;
	Tail2: cmp r1, r5
	bge Ret
	ldr r7, [r1], #4
	str r7, [r2], #4
	b Tail2

Ret: return (r - rsave);
};
	Ret: sub r0, r2, r6
	ldmfd sp!, {r4-r7, pc}

int sort (char** p, int dup, mode, np) {	.global sort .extern merge sort: stmfd sp!, {r4-r8, lr} @ p, dup, mode, np in r0, r1, r2, r3

char** q, r, rmax;
int n1, n2, nn;
char* tmp;
	@ q, r, rmax in r4, r5, r4 (reuse)
	@ n1, n2, nn in r6, r7, r6 (reuse)
	@ tmp in r8

if (np ≤ 1) goto Ret:	cmp r3, #1 ble Ret

n1 = np div 2;
n2 = np - n1;
q = p + n1;
	mov r6, r3, LSR #1
	sub r7, r3, r6
	add r4, r0, r6, LSL #2

n1 = sort (p, dup, mode, n1);	stmfd sp!, {r0-r3} mov r3, r6 bl sort mov r6, r0
n2 = sort (q, dup, mode, n2);	ldr r1, [sp, #4] @ partial restore ldr r2, [sp, #8] mov r0, r4 mov r3, r7 bl sort mov r7, r0 ldmfd sp!, {r0-r3}

np = n1 + n2;
r = allocate (np);
	add r3, r6, r7
	sub sp, sp, r3, LSL #2
	mov r5, sp

nn = merge (p, q, r, dup, mode, n1, n2);	stmfd sp!, {r0-r3} mov r3, r1 mov r1, r4 str r2, [sp, #-4]! mov r2, r5 str r6, [sp, #-4]! str r7, [sp, #-4]! bl merge add sp, sp, #12 mov r6, r0, LSR #2 ldmfd sp!, {r0-r3}
rmax = r + nn;	add r4, r5, r6, LSL #2

Loop: tmp = *r++;
*p++ = tmp;
if (r < rmax)
goto Loop;
	Loop: ldr r8, [r5], #4
	str r8, [r0], #4
	cmp r5, r4
	blt Loop

dispose (np);
return (nn);
	add sp, sp, r3, LSL #2
	mov r0, r6
	ldmfd sp!, {r4-r8, pc}

Ret: return (np);
};
	Ret: mov r0, r3
	ldmfd sp!, {r4-r8, pc}

```
@ test program
    .extern fgets, prints, strlen, atoi, sort
    .text
main:
    ldr r0, =prompt1
    bl prints
    ldr r0, =Mode
    mov r1, #4
    mov r2, #0
    bl fgets

    ldr r0, =prompt2
    bl prints
    ldr r0, =Dup
    mov r1, #4
    mov r2, #0
    bl fgets

    ldr r0, =prompt3
    bl prints
    ldr r0, =Numstr
    mov r1, #4
    mov r2, #0
    bl fgets
    bl atoi
    mov r4, r0

    ldr r0, =prompt4
    bl prints
    mov r3, r4
    ldr r5, =Buffer
    ldr r6, =List
Loop1: mov r0, r5
    str r5, [r6], #4
    mov r1, #16
    mov r2, #0
    bl fgets
    bl strlen
    add r5, r5, r0
    add r5, r5, #1
    subs r3, r3, #1
    bne Loop1

    ldr r0, =List
    mov r3, r4
    ldr r5, =Dup
    ldr r1, [r5]
```

```
   and r1, r1, #1
   ldr r5, =Mode
   ldr r2, [r5]
   and r2, r2, #1
   bl sort
   mov r3, r0
   ldr r0, =prompt5
   bl prints

   ldr r6, =List
Loop2: ldr r0, [r6], #4
   bl prints
   ldr r0, = Line
   bl prints
   subs r3, r3, #1
   bne Loop2

   mov r0, #0x18
   mov r1, #0
   swi 0x123456
   .data
prompt1: .asciz "Specify Mode - '0' for case sensitive, '1' for case insensitive \n"
prompt2: .asciz "Specify Dup - '0' for retention, '1' for removal \n"
prompt3: .asciz "Specify number of strings in the list (up to 20) \n"
prompt4: .asciz "Enter the strings now. \n"
prompt5: .asciz "Sorted list follows. \n"
Mode:   .word 0
Dup:    .word 0
Numstr: .word 0
List:   .space 80
Buffer: .space 320
Line:   .asciz "\n"
.end
```