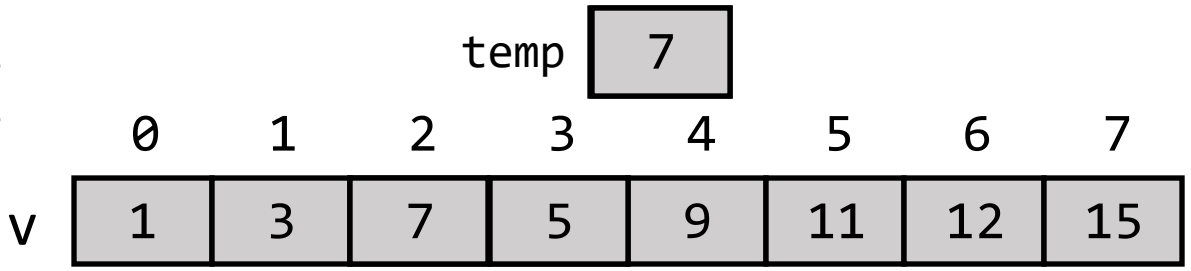
Topic V13

Bubble Sort

Reading: (Section 2.13)

C Sort Using Bubble Sort




```
void swap(int v[], int k){
      int temp;
      temp = v[k];
      v[k] = v[k+1];
      v[k+1] = temp;
}
```

```
swap(v, 2)
```

Assumption

```
   v ↔ a0
   k ↔ a1
temp ↔ t0
```

How does swap works in a processor?

C Sort Using Bubble Sort

```
void swap(int v[], int k){
      int temp;
      temp = v[k];
      v[k] = v[k+1];
      v[k+1] = temp;
}
```

swap(v, 2)

Processor

a0	0x1001 1018
a1	0x0000 0002
t0	0x0000 000B
t2	0x0000 000B

temp 7

	0	1	2	3	4	5	6	7
v	1	3	7	5	9	11	12	15

Address	Value	
0x1001 003C		
0x1001 0038		
0x1001 0034	0x0000 000F	
0x1001 0030	0x0000 000C	
0x1001 002C	0x0000 000B	
0x1001 0028	0x0000 0009	
0x1001 0024	0x0000 0005	v[k+1]
0x1001 0020	0x0000 0007	v[k]
0x1001 001C	0x0000 0003	
0x1001 0018	0x0000 0001	
0x1001 0014		
0x1001 0010		

```
void swap(int v[], int k){
      int temp;
      temp = v[k];
      v[k] = v[k+1];
      v[k+1] = temp;
}
```

Assumption

v ↔ a0

k ↔ a1

temp ↔ t0

Must separate address computation from load

```
Intermediate code (2):
temp ← M[v+k*4]
tA ← M[v+(k+1)*4]
M[v+k*4] ← tA
M[v+(k+1)*4] ← temp
```

```
Intermediate code (3):
tB ← v+k*4
temp ← M[tB]
tA ← M[v+(k+1)*4]
M[tB] ← tA
M[v+(k+1)*4] ← temp
```

Rewrite as:

v + k * 4 + 4 = tB + 4

Multiplication and addition should be two statements

Can't load from and store to memory in a single statement

```
Intermediate code (1):
temp ← M[v+k*4]
M[v+k*4] ← M[v+(k+1)*4]
M[v+(k+1)*4] ← temp
```

```
Intermediate code (4):
tB ← v+k*4
temp ← M[tB]
tA ← M[tB+4]
M[tB] ← tA
M[tB+4] ← temp
```

```
void swap(int v[], int k){
      int temp;
      temp = v[k];
      v[k] = v[k+1];
      v[k+1] = temp;
}
```

Assumption

v ↔ a0

k ↔ a1

temp ↔ t0

a0

a1

```
Intermediate code (5):
tB ← k*4
tB ← v + tB
temp ← M[tB]
tA ← M[tB+4]
M[tB] ← tA
M[tB+4] ← temp
```

```
Intermediate code (6):
tB ← a1 << 2
tB ← a0 + tB
t0 ← M[tB]
tA ← M[tB+4]
M[tB] ← tA
M[tB+4] ← t0
```

Multiplication and addition should be two statements

```
Intermediate code (4):
tB ← v+k*4
temp ← M[tB]
tA ← M[tB+4]
M[tB] ← tA
M[tB+4] ← temp
```

```
Intermediate code (7):
t1 ← a1 << 2
t1 ← a0 + t1
t0 ← M[t1]
tA ← M[t1+4]
M[t1] ← tA
M[t1+4] ← t0
```

The Swap Procedure

```
void swap(int v[], int k){
      int temp;
      temp = v[k];
      v[k] = v[k+1];
      v[k+1] = temp;
}
```

Assumption

v ↔ a0

k ↔ a1

temp ↔ t0

```
Intermediate code (7):
t1 ← a1 << 2
t1 ← a0 + t1
t0 ← M[t1]
t2 ← M[t1+4]
M[t1] ← t2
M[t1+4] ← t0
```

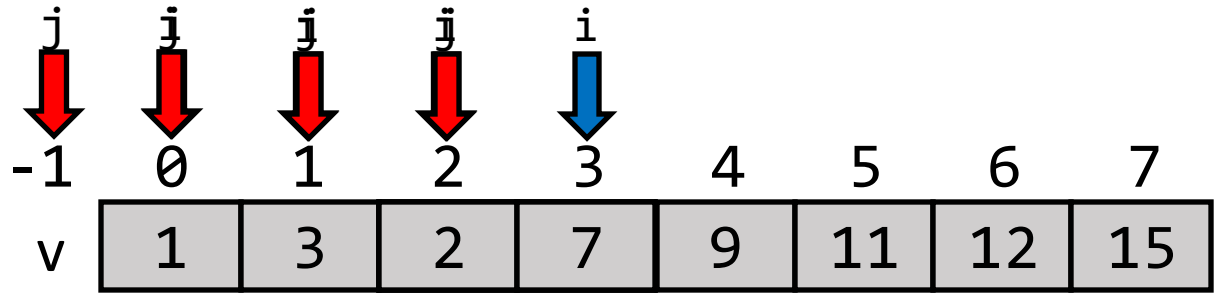
```
swap: slli  t1, a1, 2      # t1 ← k * 4
      add   t1, a0, t1     # t1 ← v + (k * 4) = address of v[k]
      lw    t0, 0(t1)      # t0 ← v[k]
      lw    t2, 4(t1)      # t2 ← v[k + 1]
      sw    t2, 0(t1)      # v[k] ← v[k + 1]
      sw    t0, 4(t1)      # v[k + 1] ← t0
      jalr  zero, ra, 0    # return to calling routine
```

The Sort Procedure

```
void sort(int v[], int n){
      int i, j;
      for (i = 0; i < n; i += 1){
            for (j = i - 1;                                    ){
                  swap(v, j);
            }
      }
}
```

Assumption

- v ↔ a0
- n ↔ a1
- i ↔ s0
- j ↔ s1

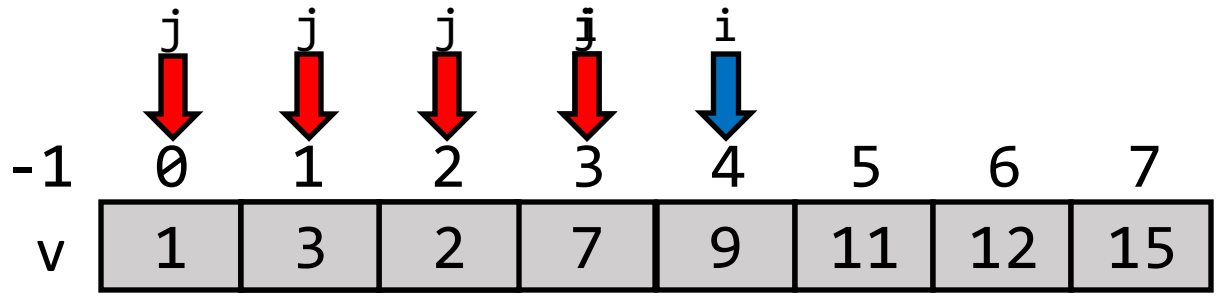


The Sort Procedure

```
void sort(int v[], int n){
      int i, j;
      for (i = 0; i < n; i += 1){
            for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1){
                  swap(v, j);
            }
      }
}
```

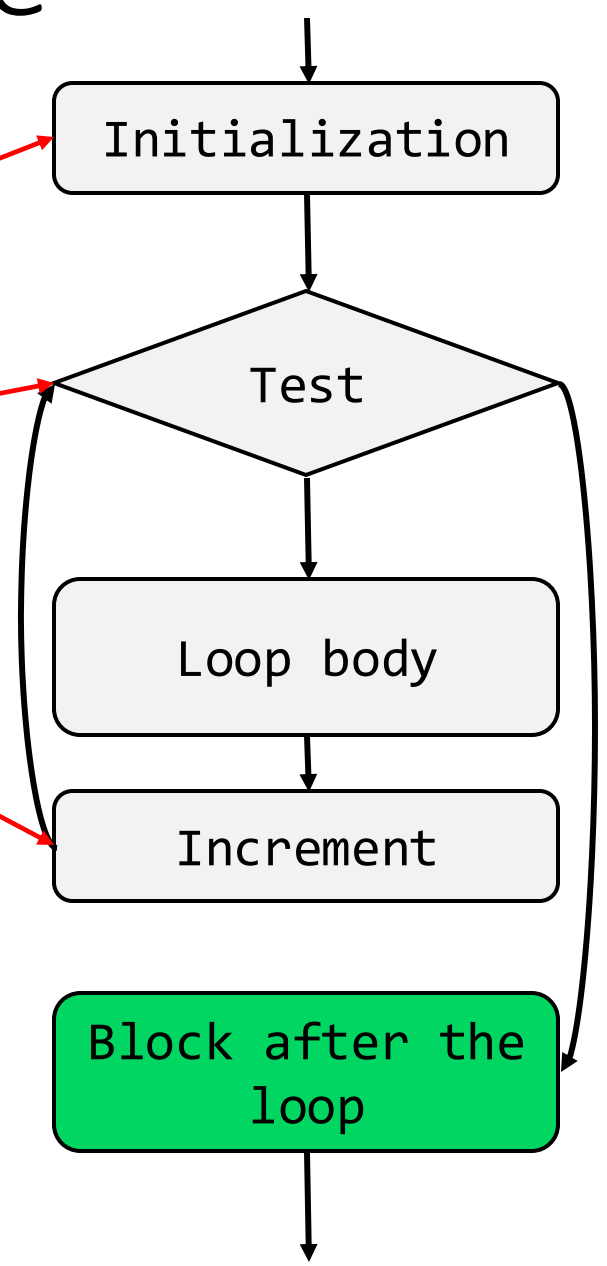
Assumption

v ↔ a0
n ↔ a1
i ↔ s0
j ↔ s1



for Loop Template

```
C code:
      …
      for (i = 0; i < n; i += 1){
            /* loop body */
      }
      /* block after the loop */
```



```
void sort(int v[], int n){
  int i, j;
  for (i = 0; i < n; i += 1){
    for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1){
      swap(v, j);
    }
  }
}
```

```
Intermediate code (1):
sort:
  i ← 0;
for1tst:
  if (i >= n) goto exit1;
  for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1){
    swap(v, j)
  }
  i ← i + 1;
  goto for1tst;
exit1:
```

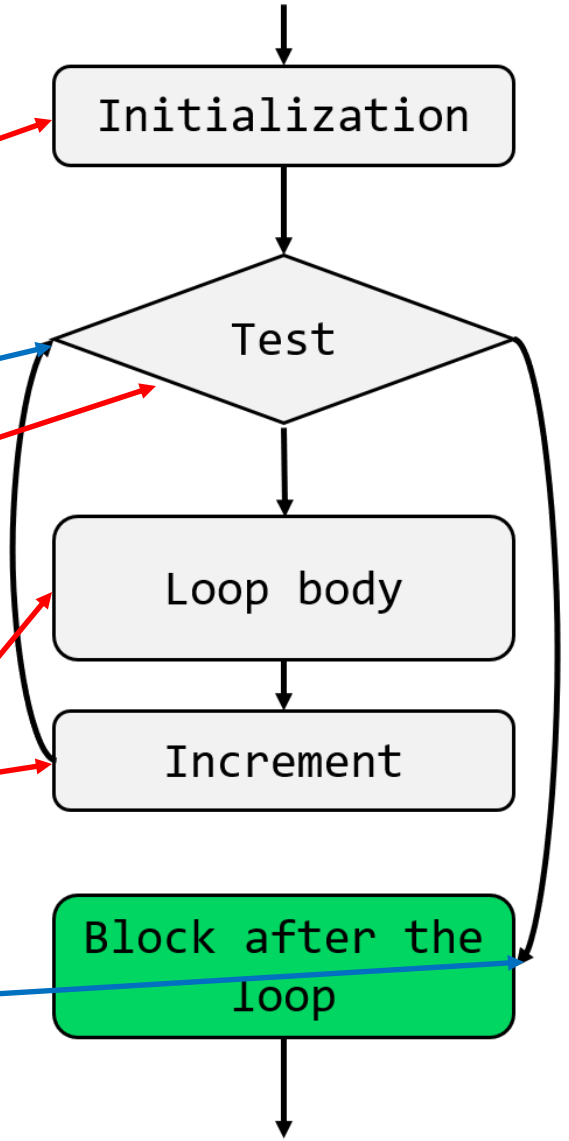
Assumption

v ↔ a0

n ↔ a1

i ↔ s0

j ↔ s1



```
void sort(int v[], int n){
  int i, j;
  for (i = 0; i < n; i += 1){
    for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1){
      swap(v, j);
    }
  }
}
```

Assumption

v ↔ a0

n ↔ a1

i ↔ s0

j ↔ s1

```
Intermediate code (1):
sort:
  i ← 0;
for1tst:
  if (i >= n) goto exit1;
  for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1){
    swap(v, j)
  }
  i ← i + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (2):
sort:
  i ← 0;
for1tst:
  if (i >= n) goto exit1;
  j ← i - 1;
for2tst:
  if (j < 0) goto exit2;
  if (v[j + 1] >= v[j]) goto exit2;
  swap(v, j);
  j ← j - 1;
  goto for2tst;
exit2:
  i ← i + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (2):
sort:
  i ← 0;
for1tst:
  if (i >= n) goto exit1;
  j ← i - 1;
for2tst:
  if (j < 0) goto exit2;
  if (v[j + 1] >= v[j]) goto exit2;
  swap(v, j);
  j ← j - 1;
  goto for2tst;
exit2:
  i ← i + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (3):
sort:
  i ← 0;
for1tst:
  if (i >= n) goto exit1;
  j ← i - 1;
for2tst:
  if (j < 0) goto exit2;
  tA ← v[j];
  tB ← v[j + 1];
  if (tB >= tA) goto exit2;
  swap(v, j);
  j ← j - 1;
  goto for2tst;
exit2:
  i ← i + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (4):
sort:
  i ← 0;
for1tst:
  if (i >= n) goto exit1;
  j ← i - 1;
for2tst:
  if (j < 0) goto exit2;
  tC ← v + 4 * j;
  tA ← M[tC];
  tB ← M[tC + 4];
  if (tB >= tA) goto exit2;
  swap(v, j);
  j ← j - 1;
  goto for2tst;
exit2:
  i ← i + 1;
  goto for1tst;
exit1:
```

Assumption

v ↔ a0
n ↔ a1
i ↔ s0
j ↔ s1

Must put v in a register that swap cannot change

And also n

```
Intermediate code (4):
sort:
  i ← 0;
for1tst:
  if (i >= n) goto exit1;
  j ← i - 1;
for2tst:
  if (j < 0) goto exit2;
  tC ← v + 4 * j;
  tA ← M[tC];
  tB ← M[tC + 4];
  if (tB >= tA) goto exit2;
  swap(v, j);
  j ← j - 1;
  goto for2tst;
exit2:
  i ← i + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (5):
sort:
  i ← 0;
for1tst:
  if (i >= n) goto exit1;
  j ← i - 1;
for2tst:
  if (j < 0) goto exit2;
  tD ← 4 * j;
  tC ← v + tD;
  tA ← M[tC];
  tB ← M[tC + 4];
  if (tB >= tA) goto exit2;
  swap(v, j);
  j ← j - 1;
  goto for2tst;
exit2:
  i ← i + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (6):
sort:
  sA ← a0; # v
  sB ← a1; # n
  i ← 0;
for1tst:
  if (i >= sB) goto exit1;
  j ← i - 1;
for2tst:
  if (j < 0) goto exit2;
  tD ← 4 * j;
  tC ← sA + tD;
  tA ← M[tC];
  tB ← M[tC + 4];
  if (tB >= tA) goto exit2;
  a0 ← sA;
  a1 ← j;
  swap(a0, a1);
  j ← j - 1;
  goto for2tst;
exit2:
  i ← i + 1;
  goto for1tst;
exit1:
```

Assumption

v ↔ a0
n ↔ a1
i ↔ s0
j ↔ s1

Intermediate code (6):

```
sort:
  sA ← a0; # v
  sB ← a1; # n
  i ← 0;
for1tst:
  if (i >= sB) goto exit1;
  j ← i - 1;
for2tst:
  if (j < 0) goto exit2;
  tD ← 4 * j;
  tC ← sA + tD;
  tA ← M[tC];
  tB ← M[tC + 4];
  if (tB >= tA) goto exit2;
  a0 ← sA;
  a1 ← j;
  swap(a0, a1);
  j ← j - 1;
  goto for2tst;
exit2:
  i ← i + 1;
  goto for1tst;
exit1:
```

Intermediate code (7):

```
sort:
  sA ← a0; # v
  sB ← a1; # n
  s0 ← 0;
for1tst:
  if (s0 >= sB) goto exit1;
  j ← s0 - 1;
for2tst:
  if (j < 0) goto exit2;
  tD ← 4 * j;
  tC ← sA + tD;
  tA ← M[tC];
  tB ← M[tC + 4];
  if (tB >= tA) goto exit2;
  a0 ← sA;
  a1 ← j;
  swap(a0, a1);
  j ← j - 1;
  goto for2tst;
exit2:
  s0 ← s0 + 1;
  goto for1tst;
exit1:
```

Intermediate code (8):

```
sort:
  sA ← a0; # v
  sB ← a1; # n
  s0 ← 0;
for1tst:
  if (s0 >= sB) goto exit1;
  s1 ← s0 - 1;
for2tst:
  if (s1 < 0) goto exit2;
  tD ← 4 * s1;
  tC ← sA + tD;
  tA ← M[tC];
  tB ← M[tC + 4];
  if (tB >= tA) goto exit2;
  a0 ← sA;
  a1 ← s1;
  swap(a0, a1);
  s1 ← s1 - 1;
  goto for2tst;
exit2:
  s0 ← s0 + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (8):
sort:
  sA ← a0; # v
  sB ← a1; # n
  s0 ← 0;
for1tst:
  if (s0 >= sB) goto exit1;
  s1 ← s0 - 1;
for2tst:
  if (s1 < 0) goto exit2;
  tD ← 4 * s1;
  tC ← sA + tD;
  tA ← M[tC];
  tB ← M[tC + 4];
  if (tB >= tA) goto exit2;
  a0 ← sA;
  a1 ← s1;
  swap(a0, a1);
  s1 ← s1 - 1;
  goto for2tst;
exit2:
  s0 ← s0 + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (9):
sort:
  sA ← a0; # v
  sB ← a1; # n
  s0 ← 0;
for1tst:
  if (s0 >= sB) goto exit1;
  s1 ← s0 - 1;
for2tst:
  if (s1 < 0) goto exit2;
  t1 ← 4 * s1;
  t2 ← sA + t1;
  t3 ← M[t2];
  t4 ← M[t2 + 4];
  if (t4 >= t3) goto exit2;
  a0 ← sA;
  a1 ← s1;
  swap(a0, a1);
  s1 ← s1 - 1;
  goto for2tst;
exit2:
  s0 ← s0 + 1;
  goto for1tst;
exit1:
```

```
Intermediate code (10):
sort:
  s2 ← a0; # v
  s3 ← a1; # n
  s0 ← 0;
for1tst:
  if (s0 >= s3) goto exit1;
  s1 ← s0 - 1;
for2tst:
  if (s1 < 0) goto exit2;
  t1 ← 4 * s1;
  t2 ← s2 + t1;
  t3 ← M[t2];
  t4 ← M[t2 + 4];
  if (t4 >= t3) goto exit2;
  a0 ← s2;
  a1 ← s1;
  swap(a0, a1);
  s1 ← s1 - 1;
  goto for2tst;
exit2:
  s0 ← s0 + 1;
  goto for1tst;
exit1:
```

```
sort:
        mv      s2, a0          # s2 ← a0;  v                   s2 ← v
        mv      s3, a1          # s3 ← a1;  n                   s3 ← n
        mv      s0, zero        # s0 ← 0;                       i ← 0
for1tst:
        bge     s0, s3, exit1   # if (s0 >= s3) goto exit1;     if i ≥ n goto exit1
        addi    s1, s0, -1      # s1 ← s0 – 1;                  j ← i – 1
for2tst:
        blt     s1, zero, exit2 # if (s1 < 0) goto exit2;      j < 0 goto exit2
        slli    t1, s1, 2       # t1 ← 4 * s1;                  t1 ← 4 * j
        add     t2, s2, t1      # t2 ← s2 + t1;                 t2 ← v + 4 * j
        lw      t3, 0(t2)       # t3 ← M[t2];                   t3 ← v[j]
        lw      t4, 4(t2)       # t4 ← M[t2 + 4];               t4 ← v[j + 1]
        bge     t4, t3, exit2   # if (t4 >= t3) goto exit2;     if v[j + 1] ≥ v[j] goto exit2
        mv      a0, s2          # a0 ← s2;                      1st param. of swap ← v
        mv      a1, s1          # a1 ← s1;                      2nd param. of swap ← j
        jal     ra, swap        # swap(a0, a1);                 swap(v, j)
        addi    s1, s1, -1      # s1 ← s1 – 1;                  j ← j – 1
        jal     zero, for2tst   # goto for2tst;                 goto for2tst
exit2:
        addi    s0, s0, 1       # s0 ← s0 + 1;                  i ← i +1
        jal     zero, for1tst   # goto for1tst;                 goto for1tst
exit1:
```

Intermediate code (10):

Move params

Outer loop

Inner loop

Pass params & call

Inner loop

Outer loop

```
sort:
        mv      s2, a0          # s2 ← a0;  v                   s2 ← v
        mv      s3, a1          # s3 ← a1;  n                   s3 ← n
        mv      s0, zero        # s0 ← 0;                       i ← 0
for1tst:
        bge     s0, s3, exit1   # if (s0 >= s3) goto exit1;     if i ≥ n goto exit1
        addi    s1, s0, -1      # s1 ← s0 – 1;                  j ← i – 1
for2tst:
        blt     s1, zero, exit2 # if (s1 < 0) goto exit2;      j < 0 goto exit2
        slli    t1, s1, 2       # t1 ← 4 * s1;                  t1 ← 4 * j
        add     t2, s2, t1      # t2 ← s2 + t1;                 t2 ← v + 4 * j
        lw      t3, 0(t2)       # t3 ← M[t2];                   t3 ← v[j]
        lw      t4, 4(t2)       # t4 ← M[t2 + 4];               t4 ← v[j + 1]
        bge     t4, t3, exit2   # if (t4 >= t3) goto exit2;     if v[j + 1] ≥ v[j] goto exit2
        mv      a0, s2          # a0 ← s2;                      1st param. of swap ← v
        mv      a1, s1          # a1 ← s1;                      2nd param. of swap ← j
        jal     ra, swap        # swap(a0, a1);                 swap(v, j)
        addi    s1, s1, -1      # s1 ← s1 – 1;                  j ← j – 1
        jal     zero, for2tst   # goto for2tst;                 goto for2tst
exit2:
        addi    s0, s0, 1       # s0 ← s0 + 1;                  i ← i +1
        jal     zero, for1tst   # goto for1tst;                 goto for1tst
exit1:
```

Intermediate code (10):

Move params

Outer loop

Inner loop

Pass params & call

Inner loop

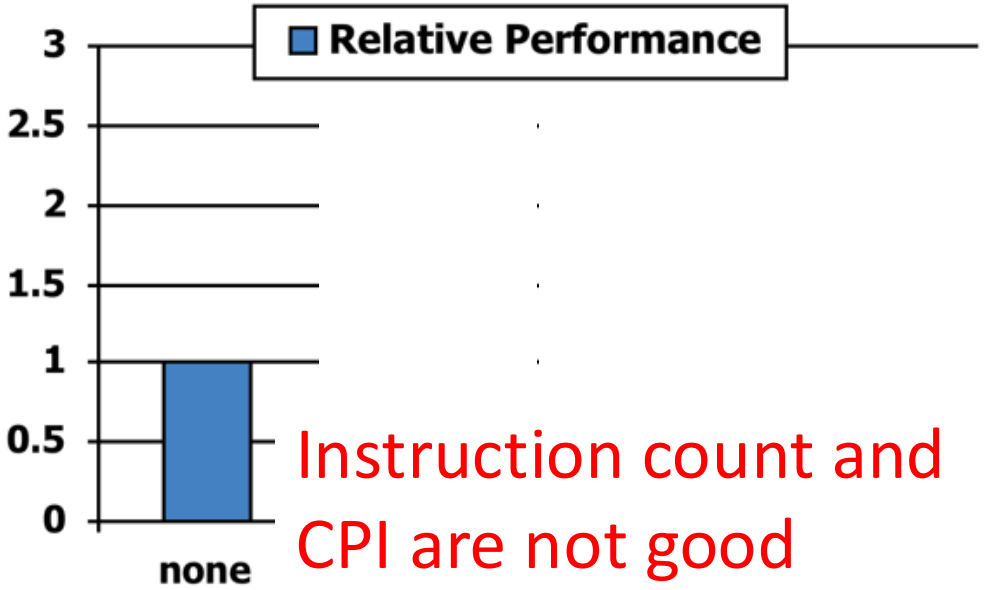
Outer loop

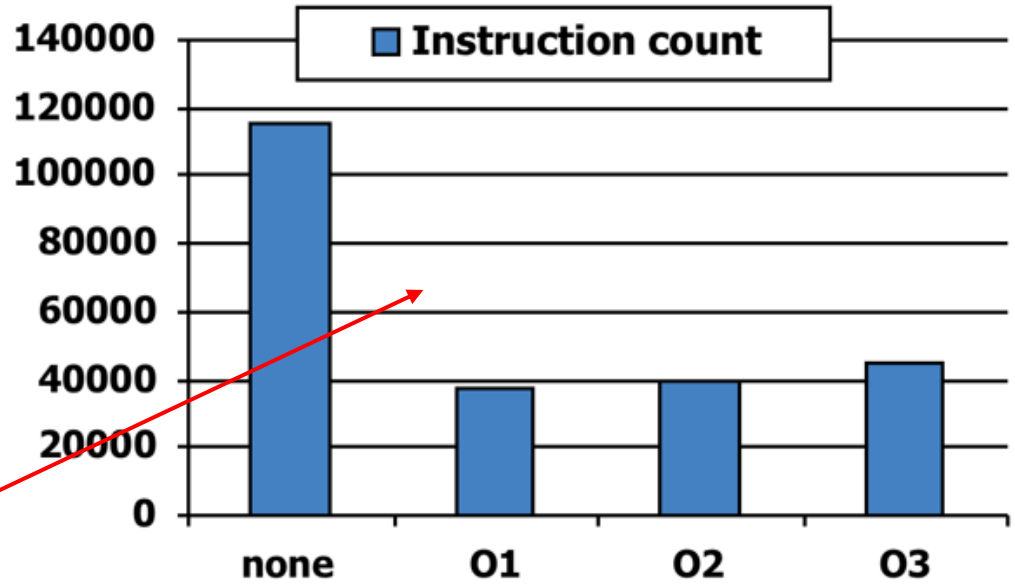
The Full Procedure

```
sort:
        addi sp, sp, -20        # make room on stack for 5 registers
        sw ra, 16(sp)           # save ra on stack
        sw s3, 12(sp)           # save s3 on stack
        sw s2, 8(sp)            # save s2 on stack
        sw s1, 4(sp)            # save s1 on stack
        sw s0, 0(sp)            # save s0 on stack

        …                       # procedure body
        …

exit1:
        lw s0, 0(sp)            # restore s0 from stack
        lw s1, 4(sp)            # restore s1 from stack
        lw s2, 8(sp)            # restore s2 from stack
        lw s3, 12(sp)           # restore s3 from stack
        lw ra, 16(sp)           # restore ra from stack
        addi sp, sp, 20         # restore stack pointer
        jalr zero, ra, 0        # return to calling routine
```

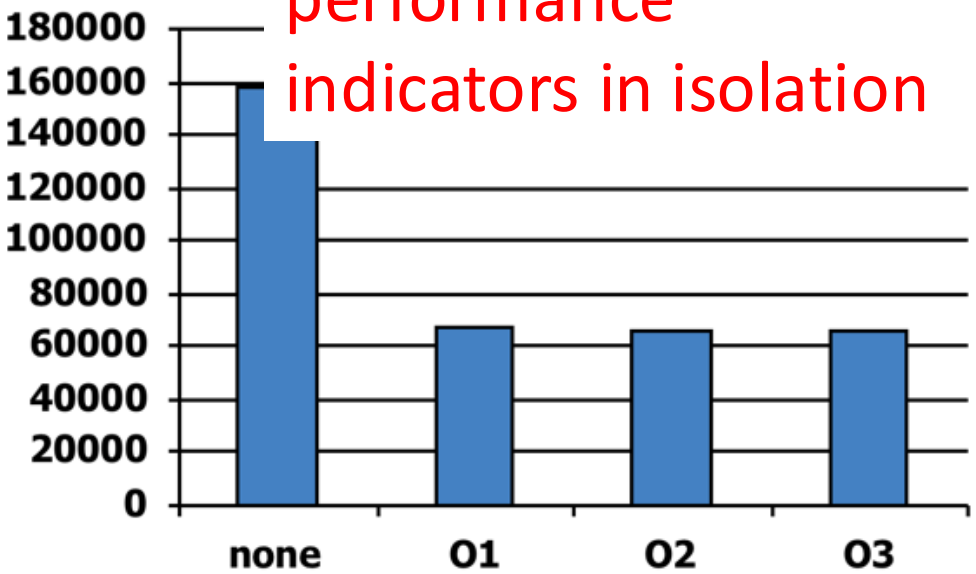
Bubble Sort Performance

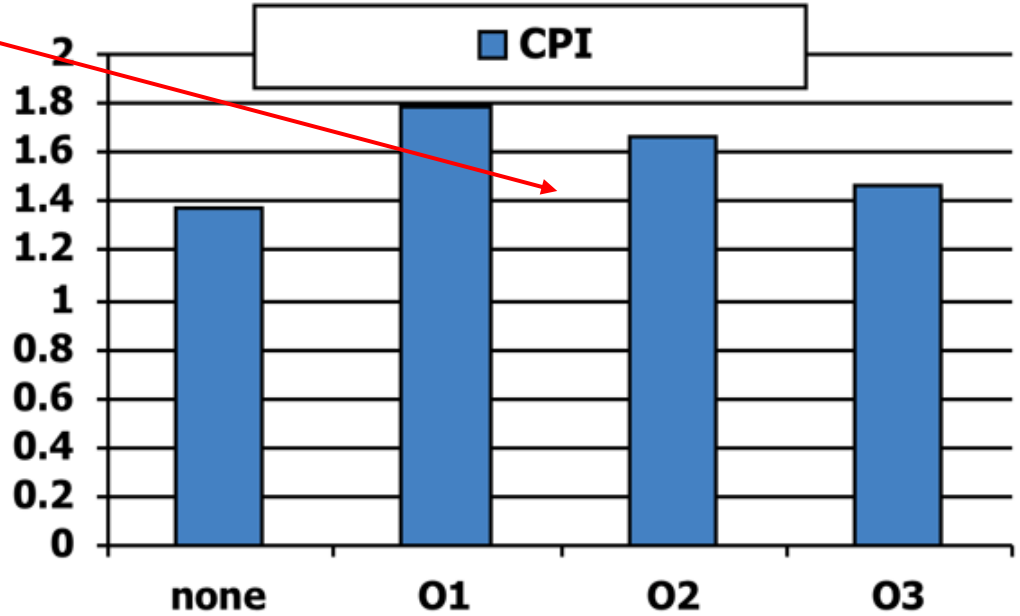
Effect of Compiler Optimization





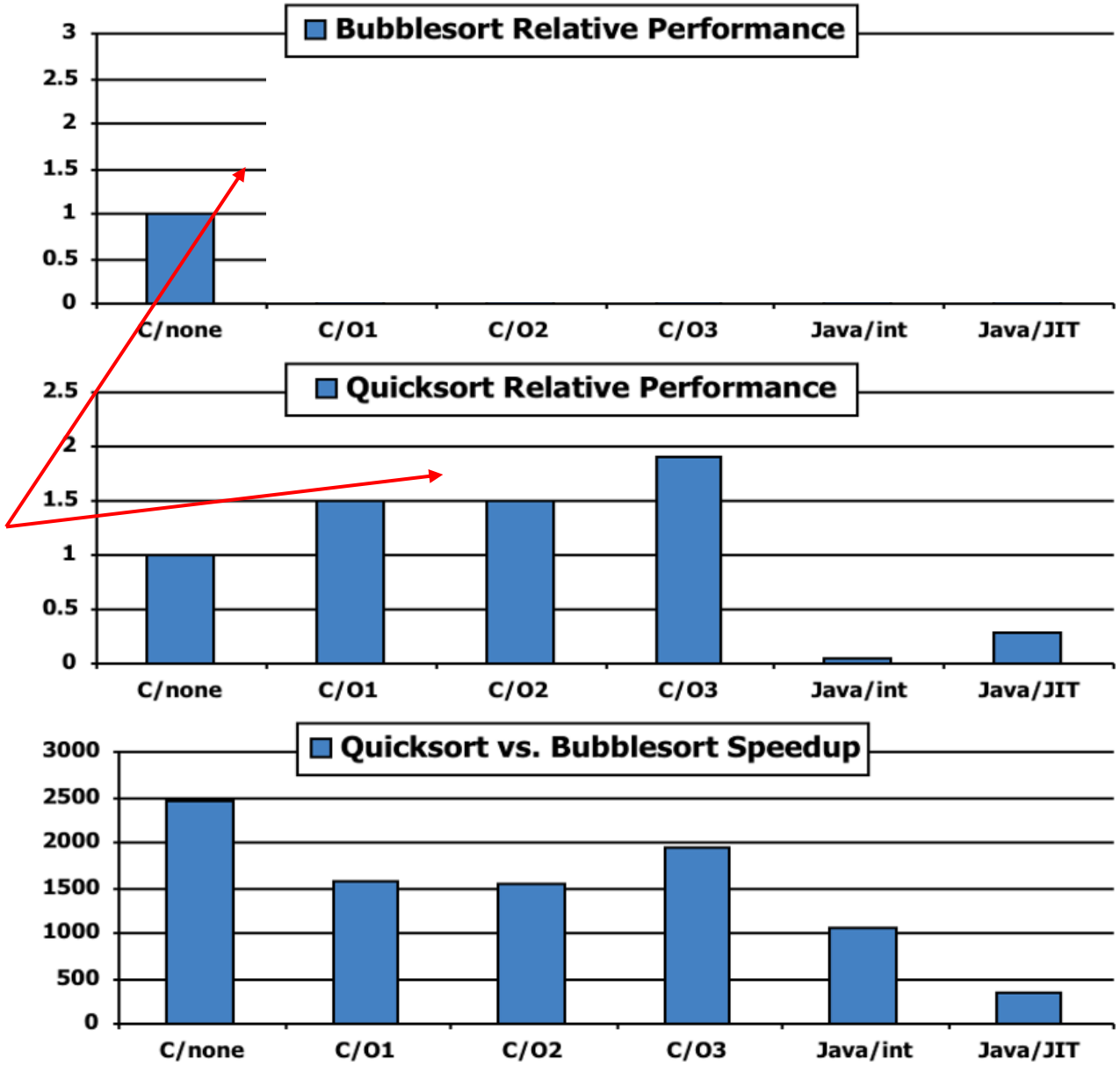
Instruction count and CPI are not good performance indicators in isolation





Compiled with gcc for Pentium 4 under Linux

Effect of Language and Algorithm



Compiler optimizations are sensitive to the algorithm

Java/JIT compiled code can be comparable to optimized C in some cases

Java/JIT compiled code is significantly faster than JVM interpreted

Nothing can fix a dumb algorithm