Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

Tutorial 06: CS31003: Compilers:

[M-07] Target Code Generation

[M-08] CFG & Local Optimizations

Indranil Sengupta
Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

isg@iitkgp.ac.in
ppd@cse.iitkgp.ac.in

October 31, 2020

Doubts from the Week

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

Functions

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

Three functions are given in the next three slides. These are to be used as inputs for the problems in this tutorial

1. Bubble Sort
2. Insertion Sort
3. Binary Search

Function 1: Bubble Sort

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems


```
void bubbleSort(int arr[], int n) {
    int i, j;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1]) {
                int t;

                t = arr[j];
                arr[j] = arr[j+1]
                arr[j+1] = t;
            }
}
```

Function 2: Insertion Sort

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems


```
void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Function 3: Binary Search

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

```
int binarySearch(int arr[], int l, int r, int x) {
    if (r >= l) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x)
            return mid;

        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        return binarySearch(arr, mid + 1, r, x);
    }
    return -1;
}
```

Problem Set 1: TAC & Peep-hole Optimization

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

For every function perform the following:

1. Convert the C function to 3 address code using our translation scheme
2. Peep-hole optimize the code

Solution: Function 1 (Bubble Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

TAC:

```
// for ( i=0; i<n-1; i++ )
// i = 0
100: t0 = 0
101: i = t0
// i < n-1
102: t1 = 1
103: t2 = n - t1
104: if i < t2 goto 109 // true exit
105: goto ... // false exit
// i++
106: t3 = i
107: i = i + 1
108: goto 102
// for ( j=0; j<n-i-1; j++ )
// j = 0
109: t4 = 0
110: j = t4
// j < n-i-1
111: t5 = n - i
112: t6 = 1
113: t7 = t5 - t6
114: if j < t7 goto 119 // true exit
115: goto 106 // false exit
// j++
116: t8 = j
117: j = j + 1
118: goto 111
// if ( arr[j] > arr[j+1] )
119: t9 = j * 4
120: t10 = arr[t9]
121: t11 = 1
122: t12 = j + t11
123: t13 = t12 * 4
124: t14 = arr[t13]
125: if t10 > t14 goto 127 // true exit
126: goto 116 // false exit
// t = arr[j]
127: t15 = j * 4
128: t16 = arr[t15]
129: t = t16
// arr[j] = arr[j+1]
130: t17 = j * 4
131: t18 = 1
132: t19 = j + t18
133: t20 = t19 * 4
134: t21 = arr[t20]
135: arr[t17] = t21
// arr[j+1] = t
136: t22 = 1
137: t23 = j + t22
138: t24 = t23 * 4
139: arr[t24] = t
140: goto 116
141: goto 106
```

Note: Explicit 'return' has not been shown for `void` return type

Solution: Function 1 (Bubble Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

Peephole optimized TAC:

```
// for ( i=0; i<n-1; i++ )
// i = 0
100:101: i = 0
// i < n-1
101:103: t2 = n - 1
102:104: if i < t2 goto 106:109 // true exit
103:105: goto ... // false exit
// i++
104:107: i = i + 1
105:108: goto 101:102
// for ( j=0; j<n-i-1; j++ )
// j = 0
106:110: j = 0
// j<n-i-1
107:111: t5 = n - i
108:113: t7 = t5 - 1
109:114: if j < t7 goto 113:119 // true exit
110:115: goto 104:106 // false exit
// j++
111:117: j = j + 1
112:118: goto 107:111
// if ( arr[j] > arr[j+1] )
113:119: t9 = j << 2
114:120: t10 = arr[t9]
115:122: t12 = j + 1
116:123: t13 = t12 << 2
117:124: t14 = arr[t13]
118:125: if t10 > t14 goto 120:127 // true exit
119:126: goto 111:116 // false exit
// t = arr[j]
120:127: t15 = j << 2
121:128: t16 = arr[t15]
122:129: t = t16
// arr[j+1] = arr[j]
123:130: t17 = j << 2
124:132: t19 = j + 1
125:133: t20 = t19 << 2
126:134: t21 = arr[t20]
127:135: arr[t17] = t21
// arr[j+1] = t
128:137: t23 = j + 1
129:138: t24 = t23 << 2
130:139: arr[t24] = t
131:140: goto 111:116
```

Solution: Function 2 (Insertion Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

TAC:

```
// for ( i=1; i<n; i++ )
// i = 1
100: t0 = 1
101: i = t0
// i<n
102: if i < n goto 107 // true exit
103: goto ... // false exit
// i++
104: t1 = i
105: i = i + 1
106: goto 102
// key = arr[i]
107: t2 = i * 4
108: t3 = arr[t2]
109: key = t3
// j = i-1
110: t4 = 1
111: t5 = i - t4
112: j = t5
// while ( j>=0 && arr[j]>key )
// j>=0
113: t6 = 0
114: if j >= t6 goto 116 // true exit
115: goto 130 // false exit
// arr[j] > key
116: t7 = j * 4
117: t8 = arr[t7]
118: if t8 > key goto 120 // true exit
119: goto 130 // false exit
// arr[j+1] = arr[j];
120: t9 = 1
121: t10 = j + t9
122: t11 = t10 * 4
123: t12 = j * 4
124: t13 = arr[t12]
125: arr[t11] = t13
// j = j - 1
126: t14 = 1
127: t15 = j - t14
128: j = t15
129: goto 113
// arr[j+1] = key
130: t16 = 1
131: t17 = j + t16
132: t18 = t17 * 4
133: arr[t18] = key
134: goto 104
```

Note: Explicit 'return' has not been shown for `void` return type

Solution: Function 2 (Insertion Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

Peephole optimized TAC:

```
// for ( i=1; i<n; i++ )
// i = 1
100:101: i = 1
// i<n
101:102: if i < n goto 106:108 // true exit
102:103: goto ... // false exit
// i++
103:105: i = i + 1
104:106: goto 101:102
// key = arr[i]
105:107: t2 = i << 2
106:108: t3 = arr[t2]
107:109: key = t3
// j = i-1
108:111: t5 = i - 1
109:112: j = t5
// while ( j>=0 && arr[j]>key )
// j>=0
110:114: if j >= 0 goto 112:116 // true exit
111:115: goto 124:130 // false exit
// arr[j] > key
112:116: t7 = j << 2
113:117: t8 = arr[t7]
114:118: if t8 > key goto 116:120 // true exit
115:119: goto 124:130 // false exit
// arr[j+1] = arr[j];
116:121: t10 = j + 1
117:122: t11 = t10 << 2
118:123: t12 = j << 2
119:124: t13 = arr[t12]
120:125: arr[t11] = t13
// j = j - 1
121:127: t15 = j - 1
122:128: j = t15
123:129: goto 110:113
// arr[j+1] = key
124:131: t17 = j + 1
125:132: t18 = t17 << 2
126:133: arr[t18] = key
127:134: goto 103:104
```

Solution: Function 3 (Binary Search)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

TAC:

```
// if ( r>= l )
100: if r >= l goto 102 // true exit
101: goto 132 // false exit
// mid = l + ( r-l )/2
102: t0 = r - l
103: t1 = 2
104: t2 = t0 / t1
105: t3 = l + t2
106: mid = t3
// if ( arr[mid] == x )
107: t4 = mid * 4
108: t5 = arr[t4]
109: if t5 == x goto 111 // true exit
110: goto 112 // false exit
111: return mid
// if ( arr[mid] > x )
112: t6 = mid * 4
113: t7 = arr[t6]
114: if t7 > x goto 116 // true exit
115: goto 124 // false exit
```

```
// binarySearch(arr,l,mid-1,x)
116: t8 = 1
117: t9 = mid - t8
118: param arr
119: param l
120: param t9
121: param x
122: t10 = call binarySearch,4
123: return t10
// binarySearch(arr,mid+1,r,x)
124: t11 = 1
125: t12 = mid + t11
126: param arr
127: param t12
128: param r
129: param x
130: t13 = call binarySearch,4
131: return t13
// return -1
132: t14 = 1
133: t15 = - t14
134: return t15
```

Solution: Function 3 (Binary Search)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

Peephole optimized TAC:

```
// if ( r>= l )
100:100: if r >= l goto 102:102 // true exit
101:101: goto 129:132 // false exit
// mid = l + ( r-l )/2
102:102: t0 = r - l
103:104: t2 = t0 / 2
104:105: t3 = l + t2
105:106: mid = t3
// if ( arr[mid] == x )
106:107: t4 = mid << 2
107:108: t5 = arr[t4]
108:109: if t5 == x goto 110:111 // true exit
109:110: goto 111:112 // false exit
110:111: return mid
// if ( arr[mid] > x )
111:112: t6 = mid << 2
112:113: t7 = arr[t6]
113:114: if t7 > x goto 115:116 // true exit
114:115: goto 122:124 // false exit
// arr[j] > key
// binarySearch(arr,l,mid-1,x)
115:117: t9 = mid - 1
116:118: param arr
117:119: param l
118:120: param t9
119:121: param x
120:122: t10 = call binarySearch,4
121:123: return t10
// binarySearch(arr,mid+1,r,x)
122:125: t12 = mid + 1
123:126: param arr
124:127: param t12
125:128: param r
126:129: param x
127:130: t13 = call binarySearch,4
128:131: return t13
// return -1
129:133: t15 = - 1
130:134: return t15
```

Problem Set 2: Memory Binding & Interval Graph

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

For every function, using the peep-hole optimized code in Problem Set 1, perform the following:

1. Generate the memory binding (layout of the activation record)
2. Compute the Interval Graph to find the minimum number of registers required

Solution: Function 1 (Bubble Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems


```
100: i = 0
101: t0 = n - 1
102: if i < t0 goto 106
103: goto ...
104: i = i + 1
105: goto 101
106: j = 0
107: t1 = n - i
108: t2 = t1 - 1
109: if j < t2 goto 113
110: goto 104
111: j = j + 1
112: goto 107
113: t3 = j << 2
114: t4 = arr[t3]
115: t5 = j + 1
116: t6 = t5 << 2
117: t7 = arr[t6]
118: if t4 > t7 goto 120
119: goto 111
120: t8 = j << 2
121: t9 = arr[t8]
122: t = t9
123: t10 = j << 2
124: t11 = j + 1
125: t12 = t11 << 2
126: t13 = arr[t12]
127: arr[t10] = t13
128: t14 = j + 1
129: t15 = t14 << 2
130: arr[t15] = t
131: goto 111
```

Activation Record

t14	int	temp	4	-68
t13	int	temp	4	-64
t12	int	temp	4	-60
t11	int	temp	4	-56
t10	int	temp	4	-52
t9	int	temp	4	-48
t8	int	temp	4	-44
t7	int	temp	4	-40
t6	int	temp	4	-36
t5	int	temp	4	-32
t4	int	temp	4	-28
t3	int	temp	4	-24
t2	int	temp	4	-20
t1	int	temp	4	-16
t0	int	temp	4	-12
j	int	local	4	-8
i	int	local	4	-4
t	int	local	4	-4
arr	int[]	param	4	+8
n	int	param	4	+12

Note: The temporary variables have been renumbered after peep-hole

Solution: Function 1 (Bubble Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

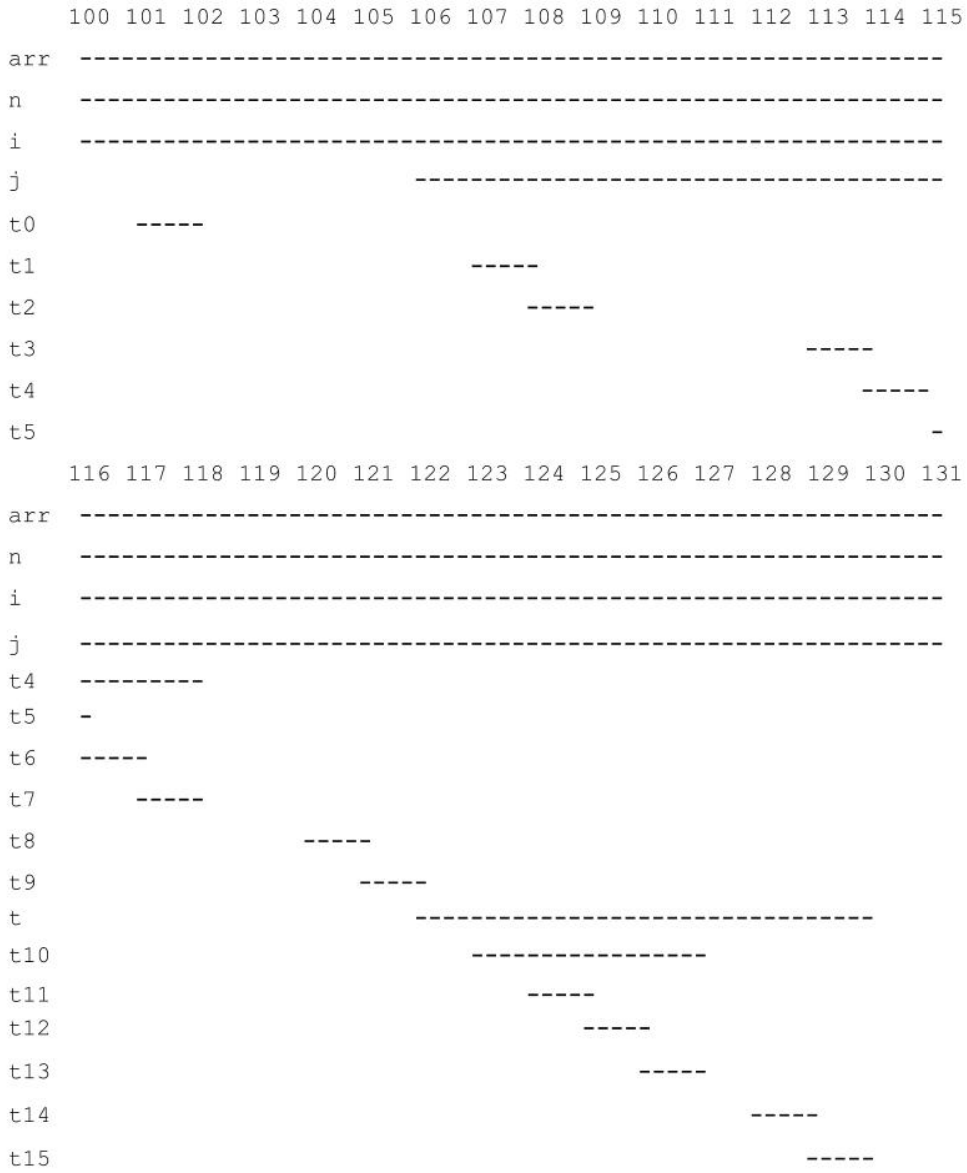
Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems



Minimum 8 registers will be needed for instants like 125 and 126

Solution: Function 2 (Insertion Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems


```
100: i = 1
101: if i < n goto 105
102: goto ...
103: i = i + 1
104: goto 101
105: t0 = i << 2
106: t1 = arr[t0]
107: key = t1
108: t2 = i - 1
109: j = t2
110: if j >= 0 goto 112
111: goto 124
112: t3 = j << 2
113: t4 = arr[t3]
114: if t4 > key goto 116
115: goto 124
116: t5 = j + 1
117: t6 = t5 << 2
118: t7 = j << 2
119: t8 = arr[t7]
120: arr[t6] = t8
121: t9 = j - 1
122: j = t9
123: goto 110
124: t10 = j + 1
125: t11 = t10 << 2
126: arr[t11] = key
127: goto 103
```

Activation Record

t11	int	temp	4	-60
t10	int	temp	4	-56
t9	int	temp	4	-52
t8	int	temp	4	-48
t7	int	temp	4	-44
t6	int	temp	4	-40
t5	int	temp	4	-36
t4	int	temp	4	-32
t3	int	temp	4	-28
t2	int	temp	4	-24
t1	int	temp	4	-20
t0	int	temp	4	-16
j	int	local	4	-12
key	int	local	4	-8
i	int	local	4	-4
arr	int[]	param	4	+8
n	int	param	4	+12

Note: The temporary variables have been renumbered after peep-hole

Solution: Function 2 (Insertion Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

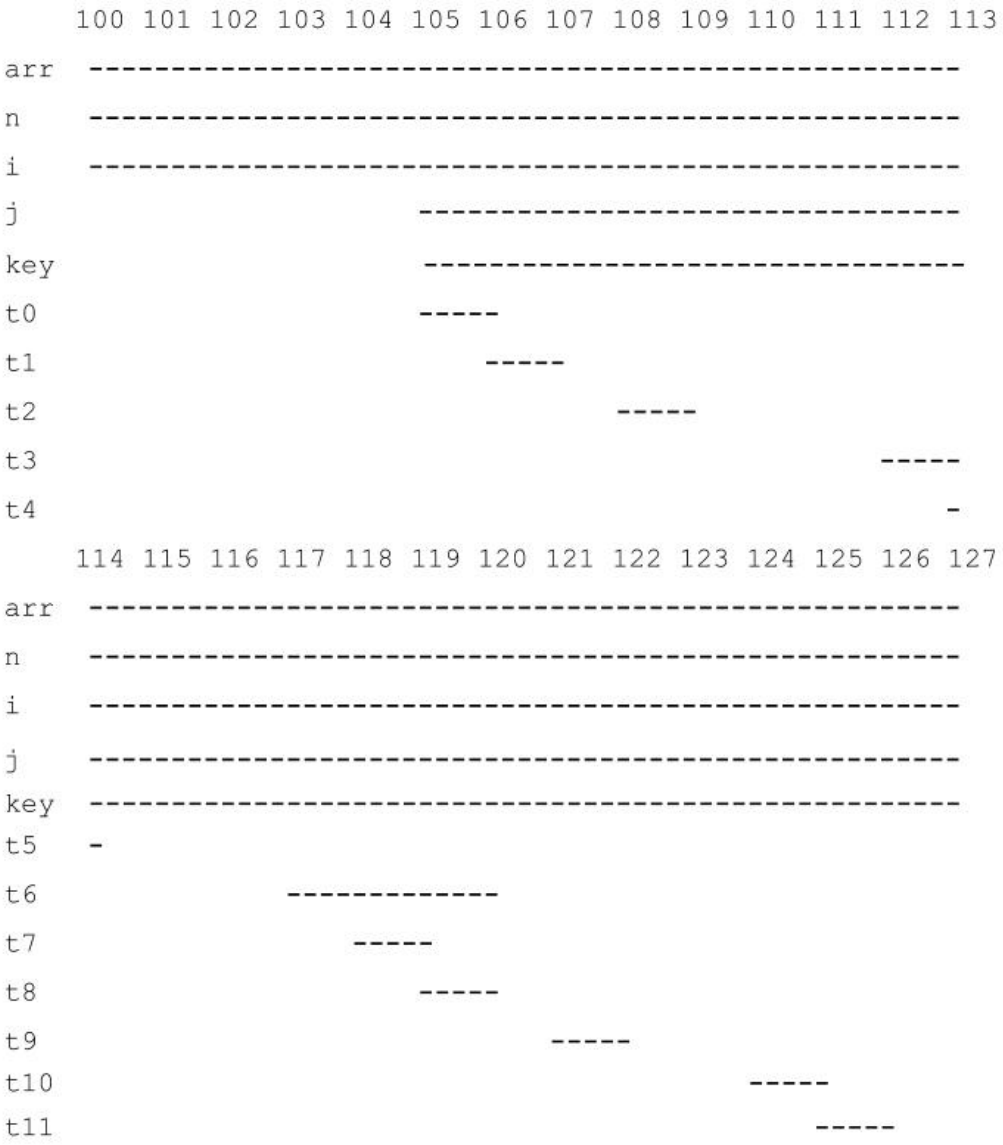
Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems



Minimum 8 registers will be needed for instant 119

Solution: Function 3 (Binary Search)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems


```
100: if r >= l goto 102
101: goto 129
102: t0 = r - l
103: t1 = t0 / 2
104: t2 = l + t1
105: mid = t2
106: t3 = mid << 2
107: t4 = arr[t3]
108: if t4 == x goto 110
109: goto 111
110: return mid
111: t5 = mid << 2
112: t6 = arr[t5]
113: if t6 > x goto 115
114 goto 122
115: t7 = mid - 1
116: param arr
117: param l
118: param t7
119: param x
120: t8 = call binarySearch,4
121: return t8
122: t9 = mid + 1
123: param arr
124: param t9
125: param r
126: param x
127: t10 = call binarySearch,4
128: return t10
129: t11 = - 1
130: return t11
```

Activation Record

t11	int	temp	4	-52
t10	int	temp	4	-48
t9	int	temp	4	-44
t8	int	temp	4	-40
t7	int	temp	4	-36
t6	int	temp	4	-32
t5	int	temp	4	-28
t4	int	temp	4	-24
t3	int	temp	4	-20
t2	int	temp	4	-16
t1	int	temp	4	-12
t0	int	temp	4	-8
mid	int	local	4	-4
arr	int[]	param	4	+8
l	int	param	4	+12
r	int	param	4	+16
x	int	param	4	+20

Note: The temporary variables have been renumbered after peep-hole

Solution: Function 3 (Binary Search)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

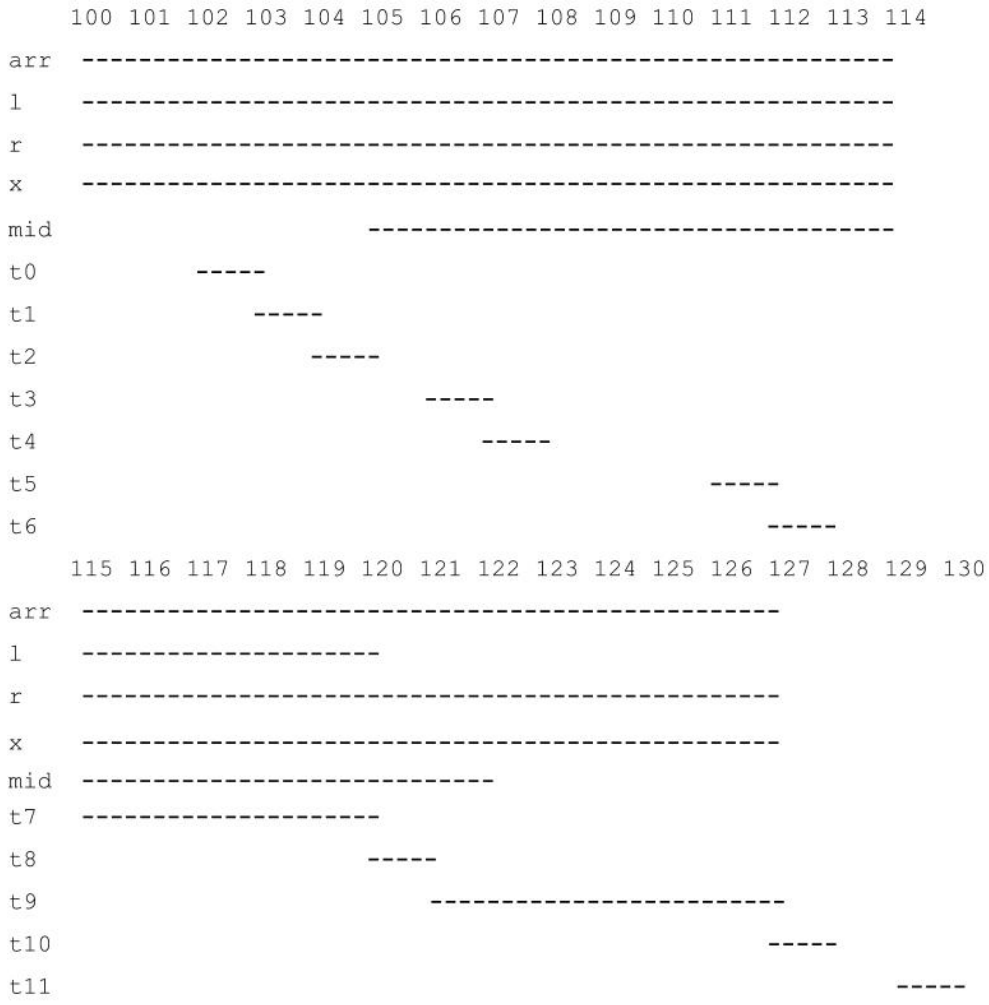
Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems



Minimum 7 registers will be needed for instants like 107, 112, 120

Problem Set 3: CFG & LCSE

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

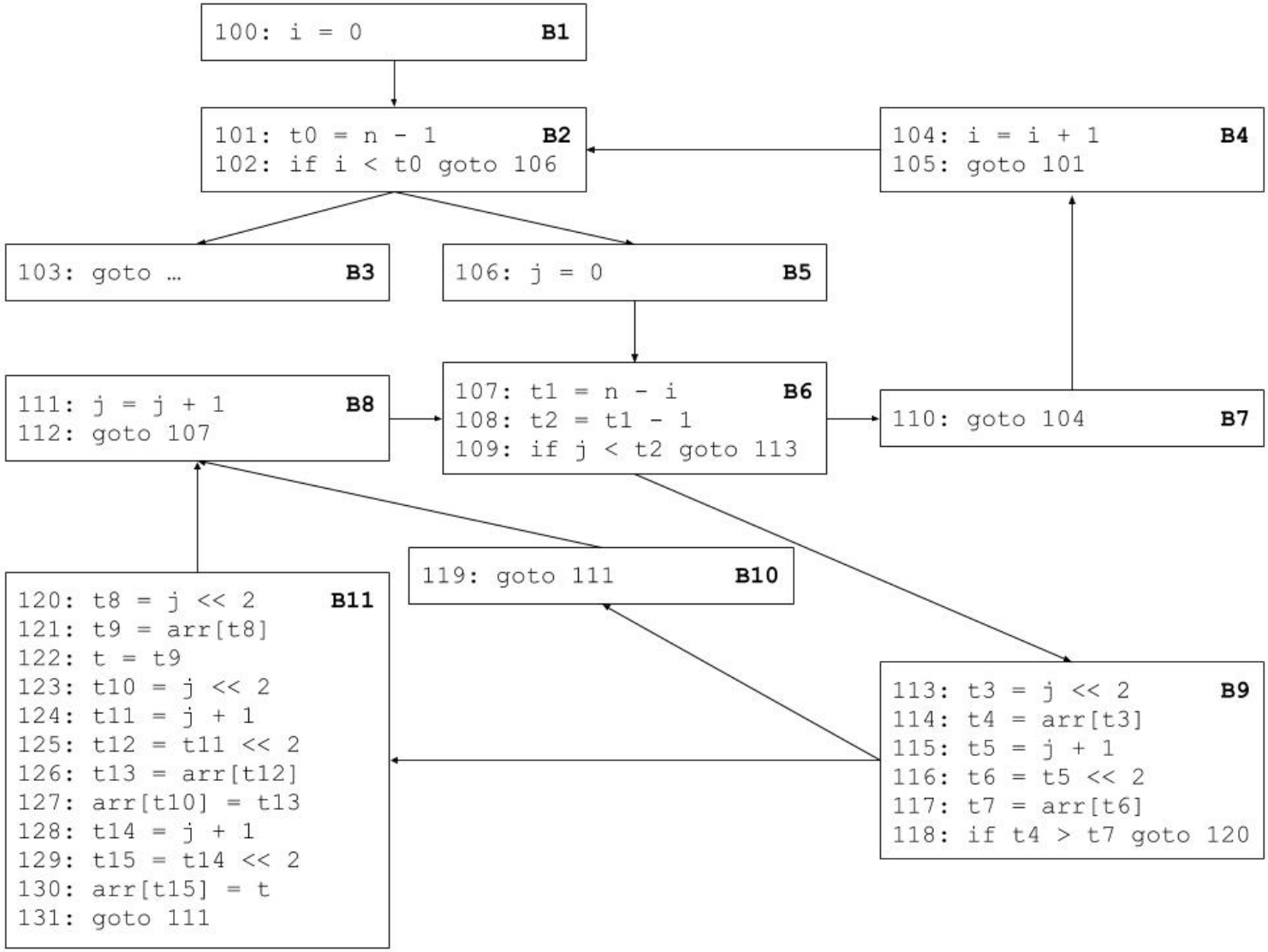
Binary Search

Practice Problems

For every function, using the peep-hole optimized code in Problem Set 1, perform the following:

1. Identify the basic blocks and draw the CFG. Remove unreacheable quad/s (not assigned to any basic block), if any
2. Use value-numbering to eliminate common sub-expressions within each block having CSE's

Solution: Function 1 (Bubble Sort)



Solution: Function 1 (Bubble Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

Blocks	VN Table		Name Table			Hash Table	
	Name	VN	Index	Name	Val	Expr	VN
B11	t8	1	1	t8, t10		j << 2	1
	t9	2	2	t9, y		j+1	3
	t	2	3	t11, t14		t11 << 2	4
	t10	1	4	t12, t15			
	t11	3	5	t13			
	t12	4					
	t13	5					
	t14	3					
	t15	4					

```
B1 : i = 0
B2 : t0 = n - 1
   : if i < t0 goto B5
B3 : goto ...
B4 : i = i + 1
   : goto B2
B5 : j = 0
B6 : t1 = n - i
   : t2 = t1 - 1
   : if j < t2 goto B9
B7 : goto B4
B8 : j = j + 1
   : goto B6
B9 : t3 = j << 2
   : t4 = arr[t3]
   : t5 = j + 1
   : t6 = t5 << 2
   : t7 = arr[t6]
   : if t4 > t7 goto B11
B10: goto B8
B11: t8 = j << 2
   : t9 = arr[t8]
   : t = t9
   // t10 = t8 XXX
   : t11 = j + 1
   : t12 = t11 << 2
   : t13 = arr[t12]
   : arr[t8] = t13
   // t14 = t11 XXX
   // t12 = t11 << 2 XXX
   : arr[t12] = t
   : goto B8
```

No changes in any other block except B11

Solution: Function 2 (Insertion Sort)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

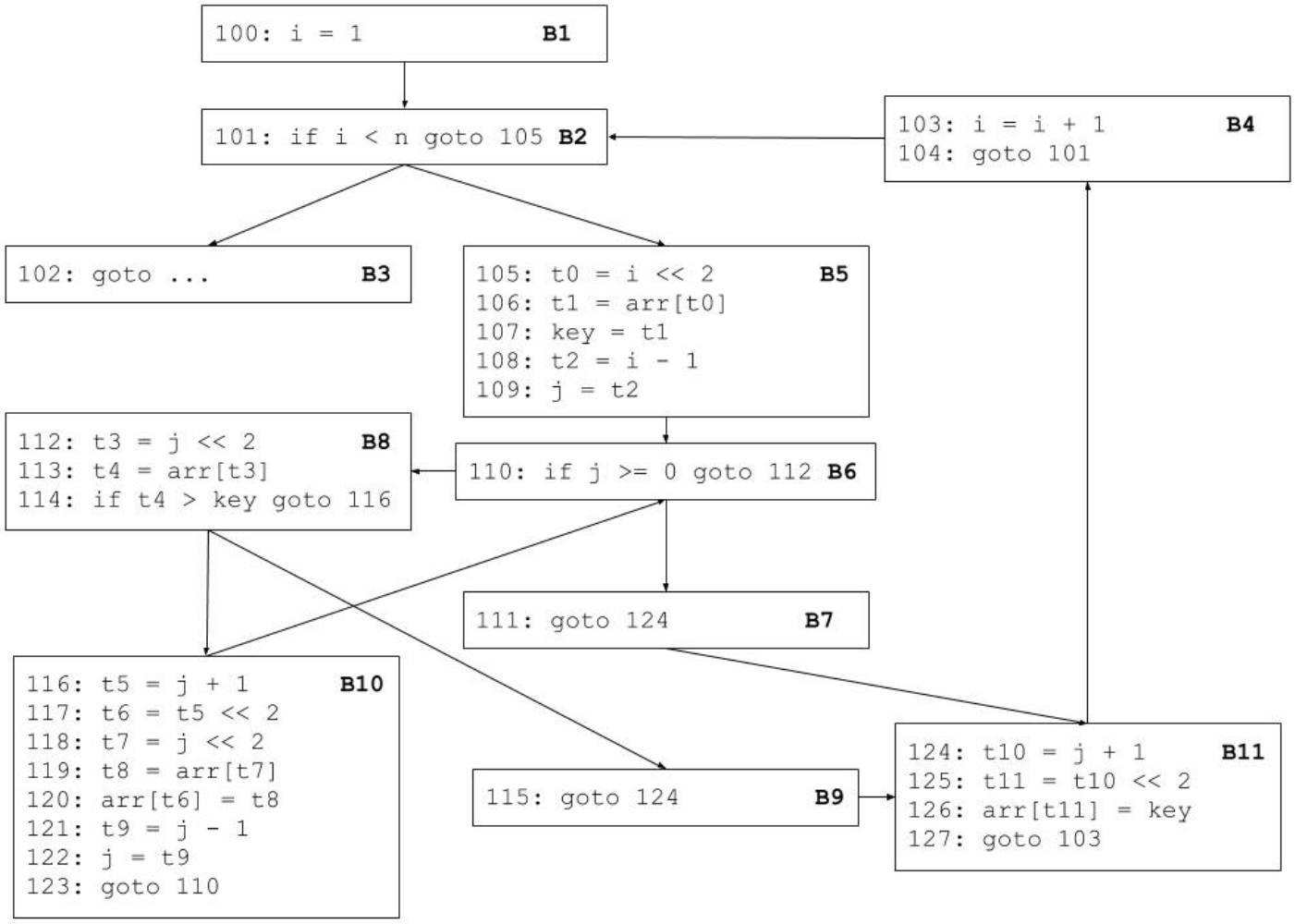
Functions
Bubble Sort
Insertion Sort
Binary Search

Peep-hole
Bubble Sort
Insertion Sort
Binary Search

Binding
Bubble Sort
Insertion Sort
Binary Search

Local CSE
Bubble Sort
Insertion Sort
Binary Search

Practice Problems



No local CSE optimizations possible within any block

Solution: Function 3 (Binary Search)

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

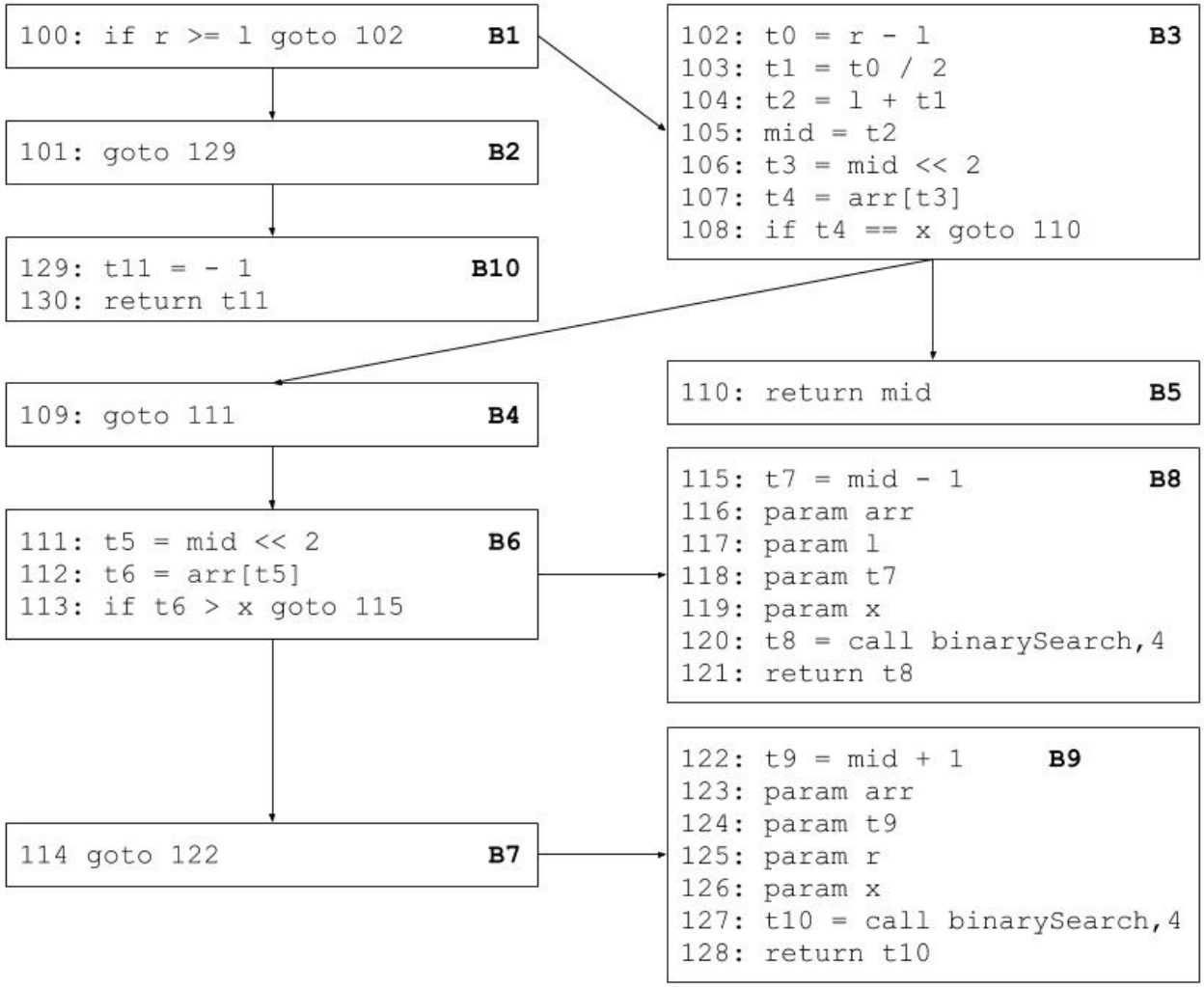
Functions
Bubble Sort
Insertion Sort
Binary Search

Peep-hole
Bubble Sort
Insertion Sort
Binary Search

Binding
Bubble Sort
Insertion Sort
Binary Search

Local CSE
Bubble Sort
Insertion Sort
Binary Search

Practice Problems



No local CSE optimizations possible within any block

Practice Problems

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions

Bubble Sort

Insertion Sort

Binary Search

Peep-hole

Bubble Sort

Insertion Sort

Binary Search

Binding

Bubble Sort

Insertion Sort

Binary Search

Local CSE

Bubble Sort

Insertion Sort

Binary Search

Practice Problems

Consider the collection of 3 function that form QuickSort:

```
void swap(int* a, int* b) {
    int t;
    t = *a;
    *a = *b;
    *b = t;
}
int partition (int arr[], int low, int high) {
    int pivot;
    pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi;
        pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

Practice Problems

Tutorial 06

I Sengupta & P P Das

Weekly Feedback

Functions
Bubble Sort
Insertion Sort
Binary Search

Peep-hole
Bubble Sort
Insertion Sort
Binary Search

Binding
Bubble Sort
Insertion Sort
Binary Search

Local CSE
Bubble Sort
Insertion Sort
Binary Search

Practice Problems

Suppose that QuickSort is invoked as:

```
quickSort(arr, 0, n - 1);
```

For every function perform the following:

1. Convert the C function to 3 address code using our translation scheme
2. Peep-hole optimize the code
3. Using the peep-hole optimized code, generate the memory binding (layout of the activation record)
4. Compute the Interval Graph to find the minimum number of registers required
5. Using the peep-hole optimized code, identify the basic blocks in the optimized code and draw the CFG. Remove unreacheable quad/s (not assigned to any basic block), if any
6. Use value-numbering to eliminate common sub-expressions within each block having CSE's