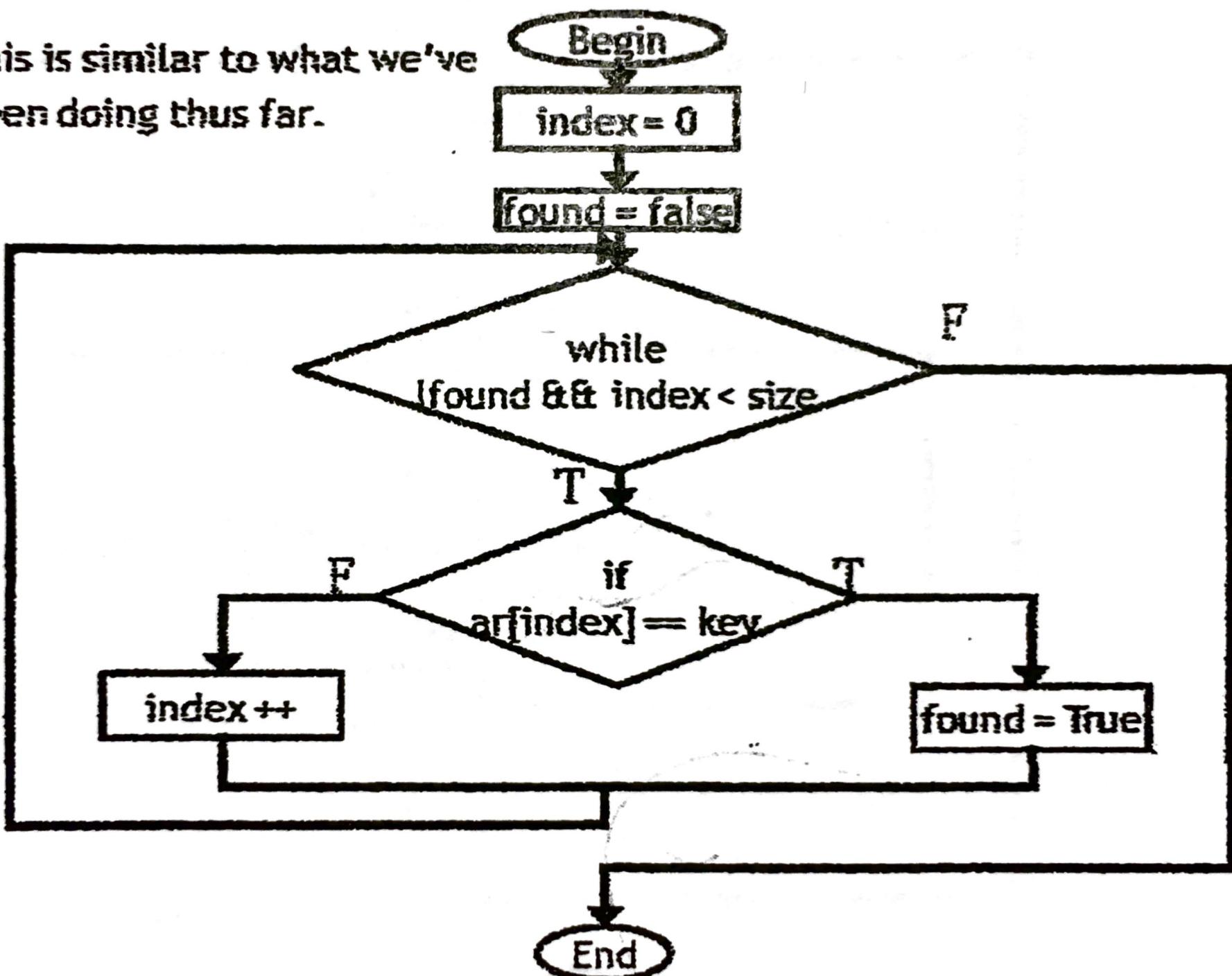


Sequential Search

- Search each element until the element is found or until you've completed the array
- Pros:
 - Easy to implement
 - Works well with unsorted arrays
- Cons: Inefficient
 - imagine an array of 100 elements

Sequential Search

This is similar to what we've been doing thus far.



in C++... How does it work?

```
const int AR_SIZE = 5;
```

```
int intAr[AR_SIZE] = { 5, 12, 6, 2, 4};
```

```
int index=0;
```

```
bool found = false;
```

```
int ikey = 6;
```

```
while (!found && index < AR_SIZE)
```

```
{
```

```
    if (intAr[index] == ikey)
```

```
{
```

```
    found = true;
```

```
}
```

```
else
```

```
{
```

```
    index ++;
```

```
}
```

index	found	ikey	intAr [0]	[1]	[2]	[3]	[4]
0	false	6	5	12	6	4	2

index	found	ikey	intAr [0]	[1]	[2]	[3]	[4]
0	false	6	5	12	6	4	2

index	found	ikey	intAr [0]	[1]	[2]	[3]	[4]
1	false	6	5	12	6	4	2

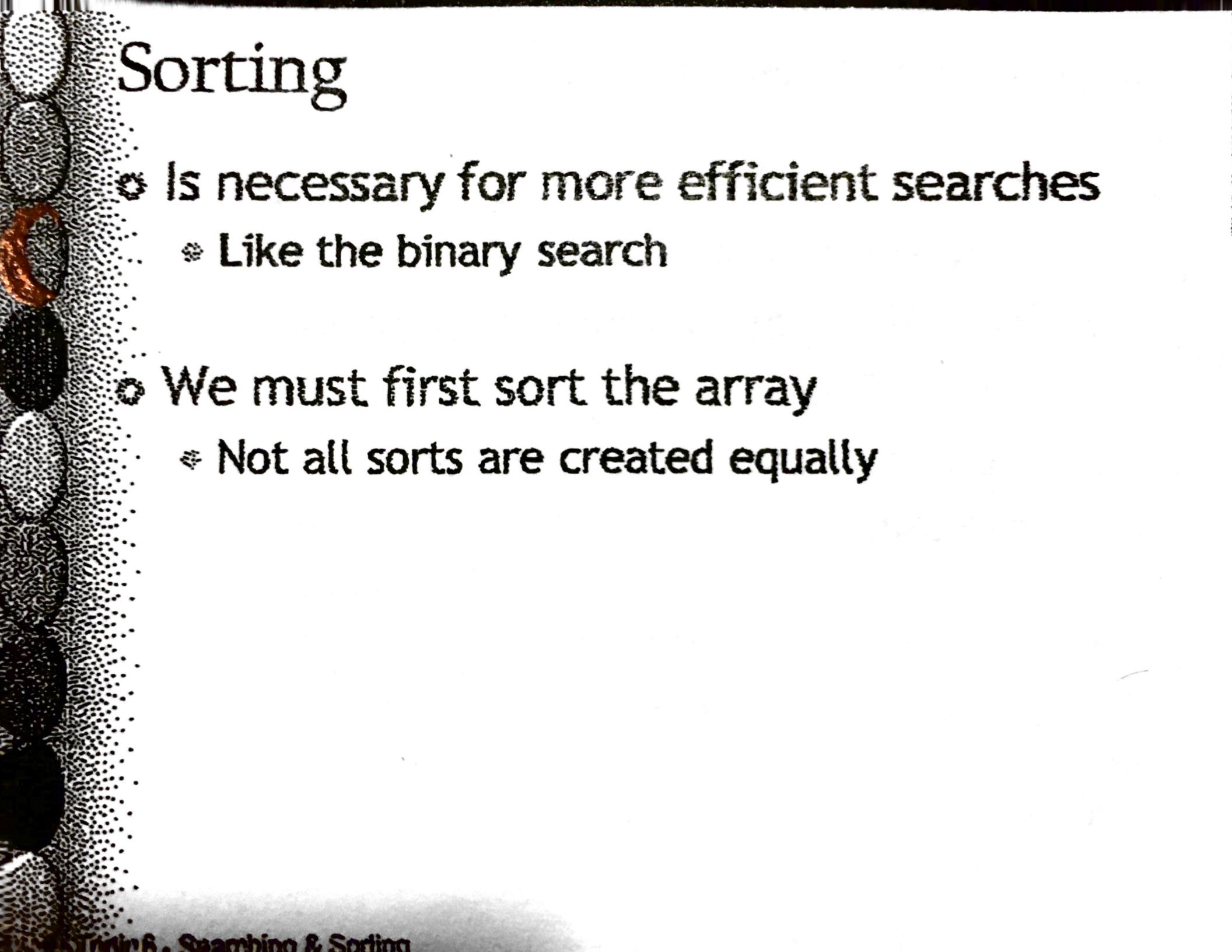
index	found	ikey	intAr [0]	[1]	[2]	[3]	[4]
2	false	6	5	12	6	4	2

index	found	ikey	intAr [0]	[1]	[2]	[3]	[4]
2	true	6	5	12	6	4	2

Completes with index = 2 → the location of the key (6)

Sorting

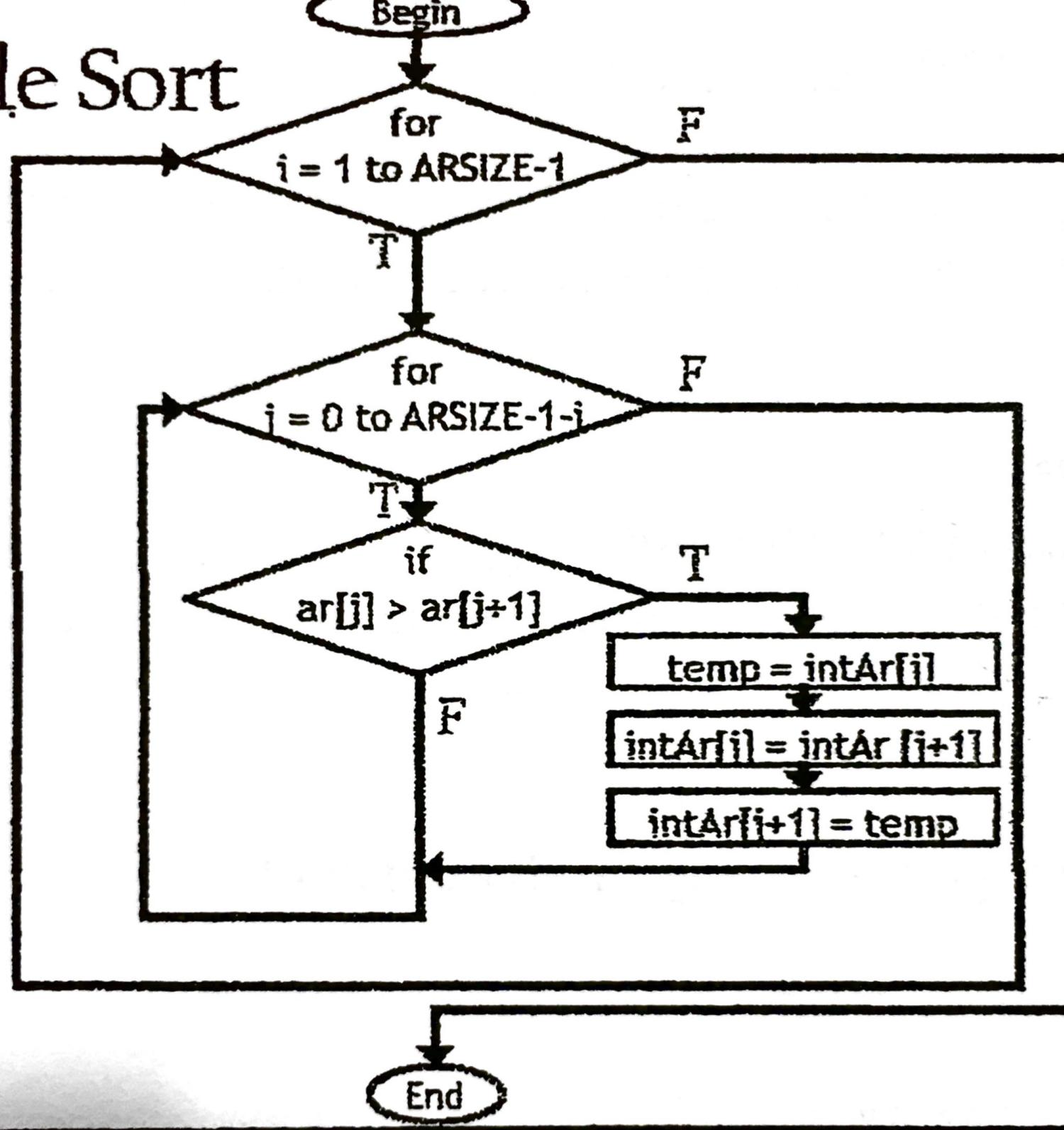
- Is necessary for more efficient searches
 - Like the binary search
- We must first sort the array
 - Not all sorts are created equally



Bubble Sort

- The bubble sort gets its name because as elements are sorted they gradually "bubble" (or rise) to their proper positions.
 - like bubbles rising in a glass of soda.
- compares adjacent elements of an
 - swaps them if they are out of order
 - repeat
- Very easy to implement BUT...
 - very inefficient → especially if the array is already in order

Bubble Sort



Bubble Sort

```
void BubbleSort(int intAr[])
```

```
{  
    int temp;  
    int i, j;  
  
    for(i = 0; i < AR_SIZE - 1; i++)  
    {  
        for(j = 0; j < AR_SIZE - 1 - i; j++)  
        {  
            if (intAr[i] > intAr[j + 1])  
            {  
                temp = intAr[i];  
                intAr[i] = intAr[j + 1];  
                intAr[j + 1] = temp;  
            }  
        }  
    }  
}
```

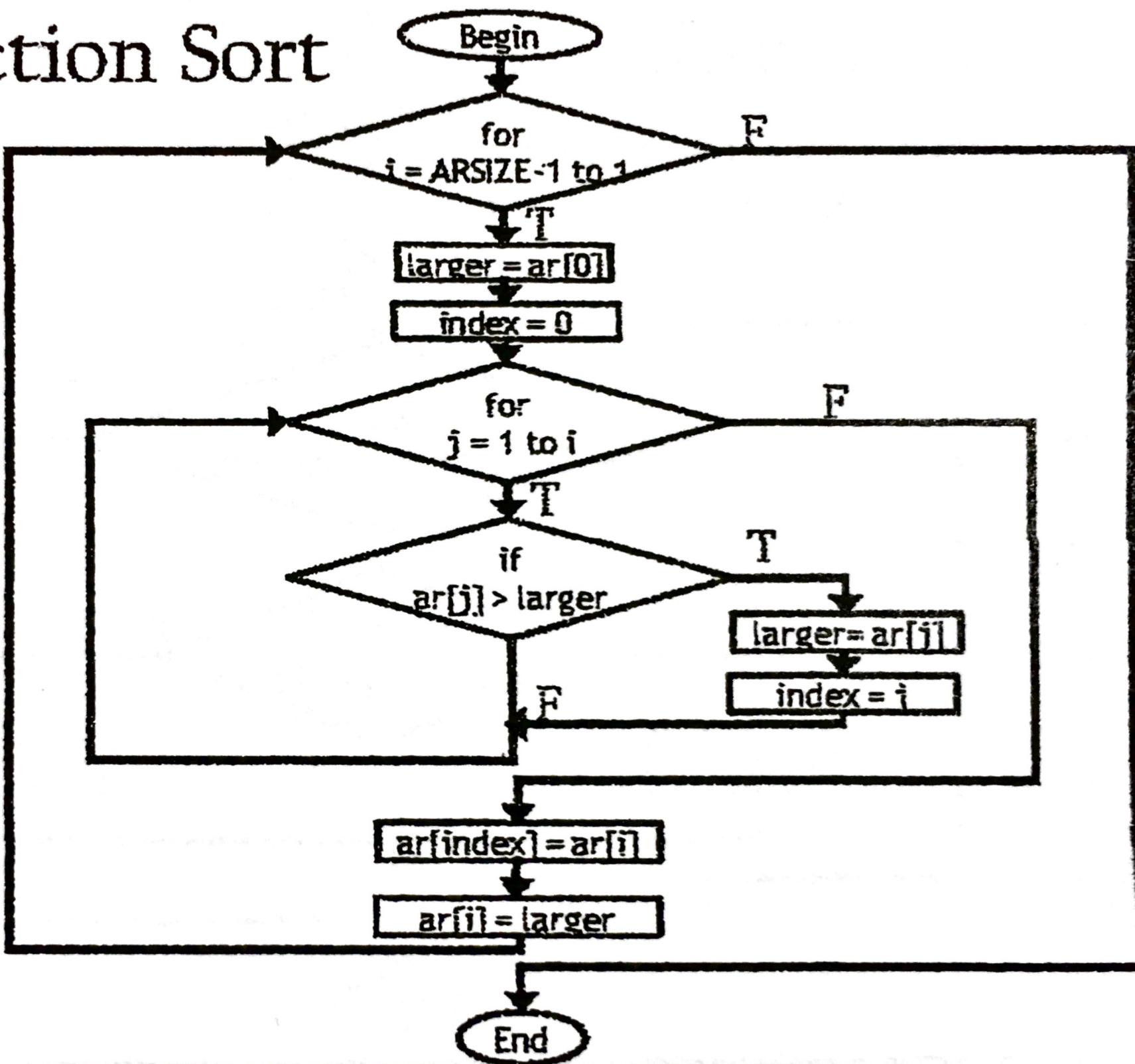
intAr	[0]	[1]	[2]	[3]	[4]
	2	4	5	6	12

i	j	temp	AR_SIZE
0	0	12	5
1	1	12	
2	2	12	
3	3	6	
4	4	6	
5	5	5	
6	6	5	
7	7	5	
8	8	4	
9	9	4	
10	10	4	
11	11	4	
12	12	4	
13	13	4	
14	14	4	
15	15	4	
16	16	4	
17	17	4	
18	18	4	
19	19	4	
20	20	4	
21	21	4	
22	22	4	
23	23	4	
24	24	4	
25	25	4	
26	26	4	
27	27	4	
28	28	4	
29	29	4	
30	30	4	
31	31	4	
32	32	4	
33	33	4	
34	34	4	
35	35	4	
36	36	4	
37	37	4	
38	38	4	
39	39	4	
40	40	4	
41	41	4	
42	42	4	
43	43	4	
44	44	4	
45	45	4	
46	46	4	
47	47	4	
48	48	4	
49	49	4	
50	50	4	
51	51	4	
52	52	4	
53	53	4	
54	54	4	
55	55	4	
56	56	4	
57	57	4	
58	58	4	
59	59	4	
60	60	4	
61	61	4	
62	62	4	
63	63	4	
64	64	4	
65	65	4	
66	66	4	
67	67	4	
68	68	4	
69	69	4	
70	70	4	
71	71	4	
72	72	4	
73	73	4	
74	74	4	
75	75	4	
76	76	4	
77	77	4	
78	78	4	
79	79	4	
80	80	4	
81	81	4	
82	82	4	
83	83	4	
84	84	4	
85	85	4	
86	86	4	
87	87	4	
88	88	4	
89	89	4	
90	90	4	
91	91	4	
92	92	4	
93	93	4	
94	94	4	
95	95	4	
96	96	4	
97	97	4	
98	98	4	
99	99	4	
100	100	4	
101	101	4	
102	102	4	
103	103	4	
104	104	4	
105	105	4	
106	106	4	
107	107	4	
108	108	4	
109	109	4	
110	110	4	
111	111	4	
112	112	4	
113	113	4	
114	114	4	
115	115	4	
116	116	4	
117	117	4	
118	118	4	
119	119	4	
120	120	4	
121	121	4	
122	122	4	
123	123	4	
124	124	4	
125	125	4	
126	126	4	
127	127	4	
128	128	4	
129	129	4	
130	130	4	
131	131	4	
132	132	4	
133	133	4	
134	134	4	
135	135	4	
136	136	4	
137	137	4	
138	138	4	
139	139	4	
140	140	4	
141	141	4	
142	142	4	
143	143	4	
144	144	4	
145	145	4	
146	146	4	
147	147	4	
148	148	4	
149	149	4	
150	150	4	
151	151	4	
152	152	4	
153	153	4	
154	154	4	
155	155	4	
156	156	4	
157	157	4	
158	158	4	
159	159	4	
160	160	4	
161	161	4	
162	162	4	
163	163	4	
164	164	4	
165	165	4	
166	166	4	
167	167	4	
168	168	4	
169	169	4	
170	170	4	
171	171	4	
172	172	4	
173	173	4	
174	174	4	
175	175	4	
176	176	4	
177	177	4	
178	178	4	
179	179	4	
180	180	4	
181	181	4	
182	182	4	
183	183	4	
184	184	4	
185	185	4	
186	186	4	
187	187	4	
188	188	4	
189	189	4	
190	190	4	
191	191	4	
192	192	4	
193	193	4	
194	194	4	
195	195	4	
196	196	4	
197	197	4	
198	198	4	
199	199	4	
200	200	4	
201	201	4	
202	202	4	
203	203	4	
204	204	4	
205	205	4	
206	206	4	
207	207	4	
208	208	4	
209	209	4	
210	210	4	
211	211	4	
212	212	4	
213	213	4	
214	214	4	
215	215	4	
216	216	4	
217	217	4	
218	218	4	
219	219	4	
220	220	4	
221	221	4	
222	222	4	
223	223	4	
224	224	4	
225	225	4	
226	226	4	
227	227	4	
228	228	4	
229	229	4	
230	230	4	
231	231	4	
232	232	4	
233	233	4	
234	234	4	
235	235	4	
236	236	4	
237	237	4	
238	238	4	
239	239	4	
240	240	4	
241	241	4	
242	242	4	
243	243	4	
244	244	4	
245	245	4	
246	246	4	
247	247	4	
248	248	4	
249	249	4	
250	250	4	
251	251	4	
252	252	4	
253	253	4	
254	254	4	
255	255	4	
256	256	4	
257	257	4	
258	258	4	
259	259	4	
260	260	4	
261	261	4	
262	262	4	
263	263	4	
264	264	4	
265	265	4	
266	266	4	
267	267	4	
268	268	4	
269	269	4	
270	270	4	
271	271	4	
272	272	4	
273	273	4	
274	274	4	
275	275	4	
276	276	4	
277	277	4	
278	278	4	
279	279	4	
280	280	4	
281	281	4	
282	282	4	
283	283	4	
284	284	4	
285	285	4	
286	286	4	
287	287	4	
288	288	4	
289	289	4	
290	290	4	
291	291	4	
292	292	4	
293	293	4	
294	294	4	
295	295	4	
296	296	4	
297	297	4	
298	298	4	
299	299	4	
300	300	4	
301	301	4	
302	302	4	
303	303	4	
304	304	4	
305	305	4	
306	306	4	
307	307	4	
308	308	4	
309	309	4	
310	310	4	
311	311	4	
312	312	4	
313	313	4	
314	314	4	
315	315	4	
316	316	4	
317	317	4	
318	318	4	
319	319	4	
320	320	4	
321	321	4	
322	322	4	
323	323	4	
324	324	4	
325	325	4	
326	326	4	
327	327	4	
328	328	4	
329	329	4	
330	330	4	
331	331	4	
332	332	4	
333	333	4	
334	334	4	
335	335	4	
336	336	4	
337	337	4	
338	338	4	
339	339	4	
340	340	4	
341	341	4	
342	342	4	
343	343	4	
344	344	4	
345	345	4	
346	346	4	
347	347	4	
348	348	4	
349	349	4	
350	350	4	
351	351	4	
352	352	4	
353	353	4	
354	354	4	</td

Selection Sort

- Finds the largest value and puts it at the end
 - * then checks the array - 1 element
 - * repeats
- Easy to implement BUT...
 - * Inefficient → imagine 1000 elements
 - * Better than the bubble sort

Selection Sort



in C++... How does it work?

```
const int AR_SIZE = 5;  
int intArray[AR_SIZE] = { 7, 12, 6, 4, 2};  
for (int i = AR_SIZE-1; i >= 1; --i)  
{  
    larger = intArray[0];  
    index = 0;  
    for (int j = 1 ; j <= i; ++j)  
    {  
        if (intArray[j] > larger)  
        {  
            larger = intArray[j];  
            index=j;  
        }  
    }  
    intArray[index] = intArray[i];  
    intArray[i] = larger;  
}
```

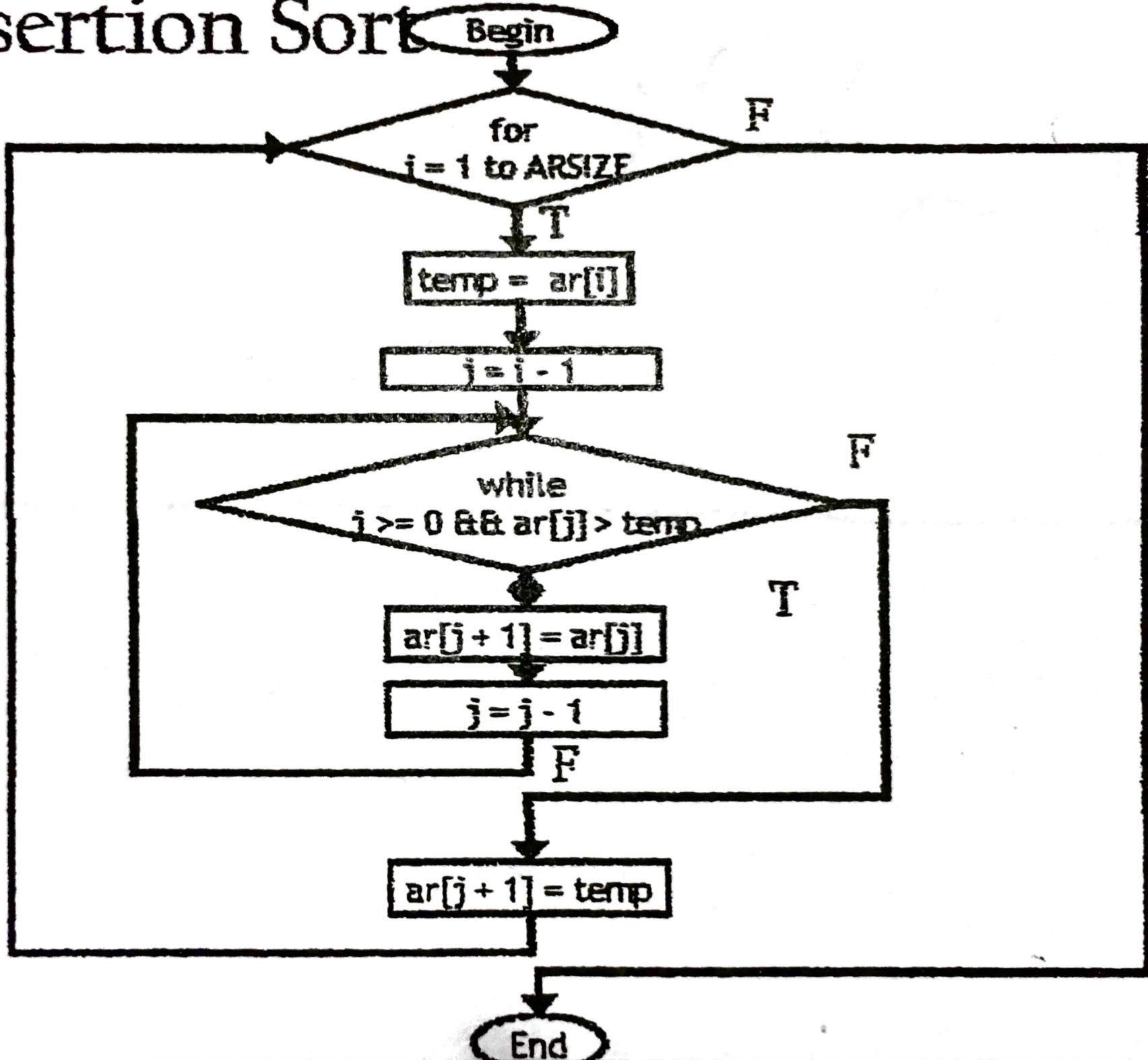
	intArray [0] [1] [2] [3] [4]							
	2	4	6	7	12			
i	4	4	3	2	1	index	larger	AR_SIZE
	3	3	3	3	3	0	7	5
	2	2	2	2	2	1	12	
	3	3	3	3	3	2	7	
	4	4	4	4	4	3	4	
	5	5	5	5	5	4	4	
	6	6	6	6	6	5	6	
	7	7	7	7	7	6	7	
	8	8	8	8	8	7	8	
	9	9	9	9	9	8	9	
	10	10	10	10	10	9	10	
	11	11	11	11	11	10	11	
	12	12	12	12	12	11	12	
	13	13	13	13	13	12	13	
	14	14	14	14	14	13	14	
	15	15	15	15	15	14	15	
	16	16	16	16	16	15	16	
	17	17	17	17	17	16	17	
	18	18	18	18	18	17	18	
	19	19	19	19	19	18	19	
	20	20	20	20	20	19	20	
	21	21	21	21	21	20	21	
	22	22	22	22	22	21	22	
	23	23	23	23	23	22	23	
	24	24	24	24	24	23	24	
	25	25	25	25	25	24	25	
	26	26	26	26	26	25	26	
	27	27	27	27	27	26	27	
	28	28	28	28	28	27	28	
	29	29	29	29	29	28	29	
	30	30	30	30	30	29	30	
	31	31	31	31	31	30	31	
	32	32	32	32	32	31	32	
	33	33	33	33	33	32	33	
	34	34	34	34	34	33	34	
	35	35	35	35	35	34	35	
	36	36	36	36	36	35	36	
	37	37	37	37	37	36	37	
	38	38	38	38	38	37	38	
	39	39	39	39	39	38	39	
	40	40	40	40	40	39	40	
	41	41	41	41	41	40	41	
	42	42	42	42	42	41	42	
	43	43	43	43	43	42	43	
	44	44	44	44	44	43	44	
	45	45	45	45	45	44	45	
	46	46	46	46	46	45	46	
	47	47	47	47	47	46	47	
	48	48	48	48	48	47	48	
	49	49	49	49	49	48	49	
	50	50	50	50	50	49	50	
	51	51	51	51	51	50	51	
	52	52	52	52	52	51	52	
	53	53	53	53	53	52	53	
	54	54	54	54	54	53	54	
	55	55	55	55	55	54	55	
	56	56	56	56	56	55	56	
	57	57	57	57	57	56	57	
	58	58	58	58	58	57	58	
	59	59	59	59	59	58	59	
	60	60	60	60	60	59	60	
	61	61	61	61	61	60	61	
	62	62	62	62	62	61	62	
	63	63	63	63	63	62	63	
	64	64	64	64	64	63	64	
	65	65	65	65	65	64	65	
	66	66	66	66	66	65	66	
	67	67	67	67	67	66	67	
	68	68	68	68	68	67	68	
	69	69	69	69	69	68	69	
	70	70	70	70	70	69	70	
	71	71	71	71	71	70	71	
	72	72	72	72	72	71	72	
	73	73	73	73	73	72	73	
	74	74	74	74	74	73	74	
	75	75	75	75	75	74	75	
	76	76	76	76	76	75	76	
	77	77	77	77	77	76	77	
	78	78	78	78	78	77	78	
	79	79	79	79	79	78	79	
	80	80	80	80	80	79	80	
	81	81	81	81	81	80	81	
	82	82	82	82	82	81	82	
	83	83	83	83	83	82	83	
	84	84	84	84	84	83	84	
	85	85	85	85	85	84	85	
	86	86	86	86	86	85	86	
	87	87	87	87	87	86	87	
	88	88	88	88	88	87	88	
	89	89	89	89	89	88	89	
	90	90	90	90	90	89	90	
	91	91	91	91	91	90	91	
	92	92	92	92	92	91	92	
	93	93	93	93	93	92	93	
	94	94	94	94	94	93	94	
	95	95	95	95	95	94	95	
	96	96	96	96	96	95	96	
	97	97	97	97	97	96	97	
	98	98	98	98	98	97	98	
	99	99	99	99	99	98	99	
	100	100	100	100	100	99	100	
	101	101	101	101	101	100	101	
	102	102	102	102	102	101	102	
	103	103	103	103	103	102	103	
	104	104	104	104	104	103	104	
	105	105	105	105	105	104	105	
	106	106	106	106	106	105	106	
	107	107	107	107	107	106	107	
	108	108	108	108	108	107	108	
	109	109	109	109	109	108	109	
	110	110	110	110	110	109	110	
	111	111	111	111	111	110	111	
	112	112	112	112	112	111	112	
	113	113	113	113	113	112	113	
	114	114	114	114	114	113	114	
	115	115	115	115	115	114	115	
	116	116	116	116	116	115	116	
	117	117	117	117	117	116	117	
	118	118	118	118	118	117	118	
	119	119	119	119	119	118	119	
	120	120	120	120	120	119	120	
	121	121	121	121	121	120	121	
	122	122	122	122	122	121	122	
	123	123	123	123	123	122	123	
	124	124	124	124	124	123	124	
	125	125	125	125	125	124	125	
	126	126	126	126	126	125	126	
	127	127	127	127	127	126	127	
	128	128	128	128	128	127	128	
	129	129	129	129	129	128	129	
	130	130	130	130	130	129	130	
	131	131	131	131	131	130	131	
	132	132	132	132	132	131	132	
	133	133	133	133	133	132	133	
	134	134	134	134	134	133	134	
	135	135	135	135	135	134	135	
	136	136	136	136	136	135	136	
	137	137	137	137	137	136	137	
	138	138	138	138	138	137	138	
	139	139	139	139	139	138	139	
	140	140	140	140	140	139	140	
	141	141	141	141	141	140	141	
	142	142	142	142	142	141	142	
	143	143	143	143	143	142	143	
	144	144	144	144	144	143	144	
	145	145	145	145	145	144	145	
	146	146	146	146	146	145	146	
	147	147	147	147	147	146	147	
	148	148	148	148	148	147	148	
	149	149	149	149	149	148	149	
	150	150	150	150	150	149	150	
	151	151	151	151	151	150	151	
	152	152	152	152	152	151	152	
	153	153	153	153	153	152	153	
	154	154	154	154	154	153	154	
	155	155	155	155	155	154	155	
	156	156	156	156	156	155	156	
	157	157	157	157	157	156	157	
	158	158	158	158	158	157	158	
	159	159	159	159	159	158	159	
	160	160	160	160	160	159	160	
	161	161	161	161	161	160	161	
	162	162	162	162	162	161	162	
	163	163	163	163	163	162	163	
	164	164	164	164	164	163	164	
	165	165	165	165	165	164	165	
	166	166	166	166	166	165	166	
	167	167	167	167	167	166	167	
	168	168	168	168	168	167	168	
	169	169						

Insertion Sort

- Passes through the array only once
 - More efficient than the selection sort
- Sorts like you would sort a hand of cards
 - pick up one card
 - put it in the correct location
 - repeat

)

Insertion Sort



in C++... How does it work?

```
const int AR_SIZE = 5;  
int intAr[AR_SIZE] = { 7, 12, 6, 14, 22};
```

```
for (int i = 1; i < AR_SIZE; ++i)
```

```
{
```

```
    temp = intAr[i];
```

```
    j = i - 1;
```

```
    while ( j >= 0 && intAr[j] > temp)
```

```
{
```

```
        intAr[ j + 1 ] = intAr[j];
```

```
        j = j - 1;
```

```
}
```

```
    intAr[ j + 1 ] = temp;
```

```
}
```

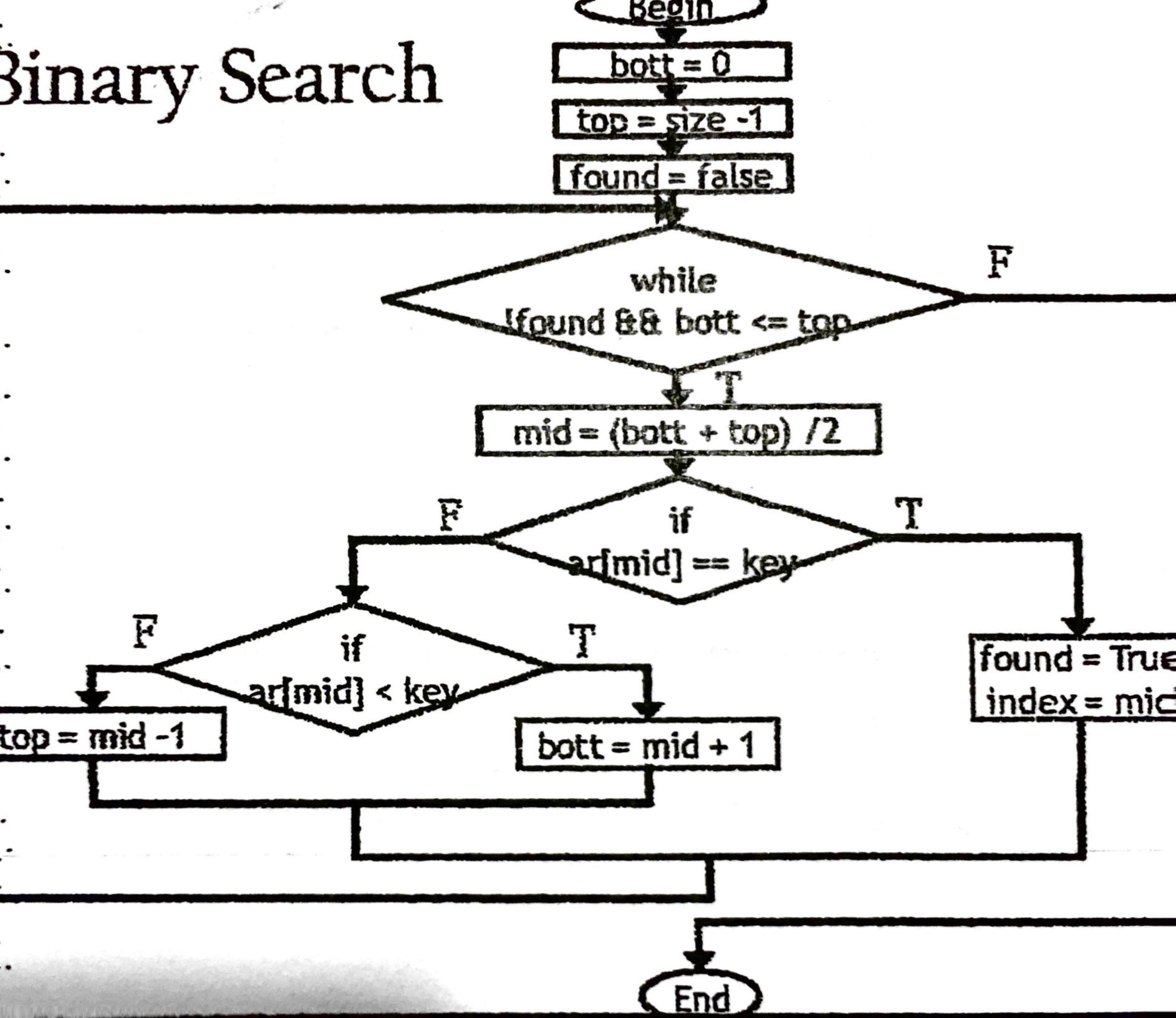
intAr	[0]	[1]	[2]	[3]	[4]
	6	7	12	14	22

i	j	temp	AR_SIZE
1	0	7	5
2	1	12	
3	2	14	
4	3	22	

Binary Search

- Now that we know how to sort..
 - We can explore better search routines
- For a Binary Search the array must be in order
- Gets its name because the algorithm continually divides the list into two parts.
- Much more efficient than the Sequential Search

Binary Search



How a Binary Search Works

0	1	2	3	4	5	6	7	8	9
2	4	12	23	24	35	41	48	51	61

Keep track of the top, bottom and middle indices

$$\text{Middle} = (\text{Top} + \text{Bottom}) / 2$$

CHECK the middle value

IF the middle value is equal to the KEY
we're done!

ELSE IF the middle value is less than the KEY
MOVE the bottom index to middle + 1

ELSE

MOVE the top index to middle - 1

How a Binary Search Works

0 1 2 3 4 5 6 7 8 9

2	4	12	23		35	41	48	51	61
---	---	----	----	--	----	----	----	----	----

- Set middle index to $(\text{top} + \text{bottom}) / 2$
- Check if bottom \leq top and if found

Middle

0 1 2 3 4 5 6 7 8 9

			35	41		51	61
--	--	--	----	----	--	----	----

Bottom is \leq top and key is not found
Key is \geq Middle so bottom = middle + 1
 $\text{middle} = (\text{top} + \text{bottom}) / 2$

Bottom

Middle

Top

0 1 2 3 4 5 6 7 8 9

41

Bottom is \leq top and key is not found
Key is $<$ Middle so top = middle - 1
 $\text{middle} = (\text{top} + \text{bottom}) / 2$

Top

0 1 2 3 4 5 6 7 8 9

41

Bottom is \leq top and key is not found
Key is \geq Middle so bottom = middle - 1
 $\text{middle} = (\text{top} + \text{bottom}) / 2$

middle = key so it's found

	Bubble	Selection	Insertion
Outer loop	For 0 → ARRsize -1	For ArrSize → 0	For 0 → Arr.size -1
inner loop	For	For	while
Task	swap element	find largest put at end	goes through each element and puts in correct order