Number of Items	(time in ms)	100,000	200,000	300,000	400,000	500,000
Insert Front	Array	**11037.6**	**44518.3**	**98831.5**	**176168**	**279474**
	Pointer	4.178	7.872	11.841	15.834	20.641
	STL	5.77	11.613	17.92	22.85	29.373
Insert Back	Array	1.134	2.089	4.024	4.113	5.292
	Pointer	**20280.5**	**84164.4**	**191806**	**341894**	**545398**
	STL	5.667	11.451	17.613	22.914	29.452
Delete Front	Array	**13697**	**54213.4**	**121528**	**216389**	**340056**
	Pointer	2.023	3.907	5.709	7.737	10.225
	STL	4.918	9.986	15.139	22.587	24.76
Delete Back	Array	0.939	1.889	2.819	3.774	4.704
	Pointer	**30343.6**	**124378**	**281079**	**508268**	**802554**
	STL	9.014	15.555	21.085	26.285	30.571
Traverse	Array	0.538	1.085	1.614	2.158	2.695
	Pointer	1.181	2.432	3.624	**6.272**	5.855
	STL	**1.359**	**2.85**	**3.912**	5.049	**6.695**
Pop	Array	0.642	1.508	2.196	2.552	3.193
	Pointer	**2.535**	**4.351**	**6.179**	**8.723**	**10.322**
	STL	1.161	2.246	3.405	4.297	5.435
Push	Array	0.945	1.687	1.927	2.814	2.746
	Pointer	**2.484**	**5.168**	**7.837**	**10.77**	**13.287**
	STL	2.188	3.021	4.561	9.516	8.132

I bolded the highest times in the table. For lists, front insertion and deletion was significantly slower with the array implementation than either of the two others, because this required all other items to be shifted. My pointer implementation was also consistently faster than the STL for this task, presumably because it has less overhead. This behavior was somewhat reversed for back insertion and deletion. Array was the fastest; my pointer implementation was very slow because to locate the end pointer the entire list had to be traversed. The STL implementation avoided this problem because it seems to have been implemented with a doubly linked list. As for stacks, all operations were relatively equal and quick, though my pointer implementation was always the slowest, I think because of the overhead of dealing with the pointers; this may be optimized in the STL implementation.